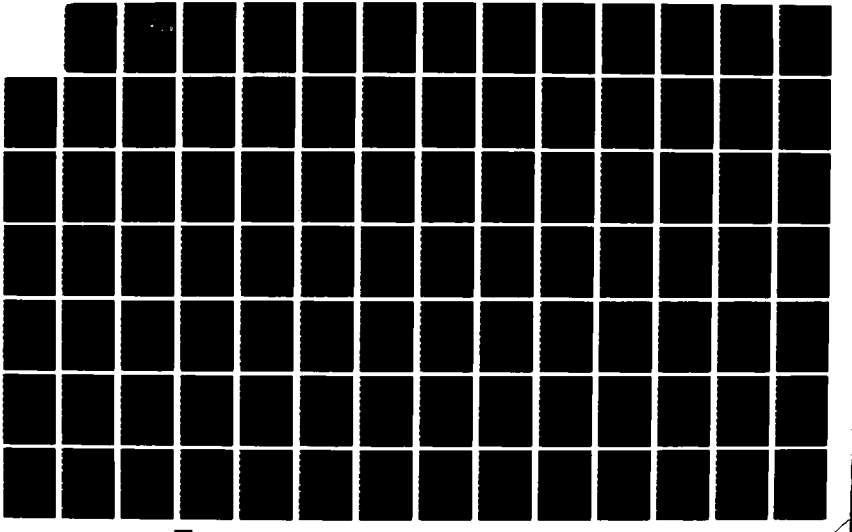
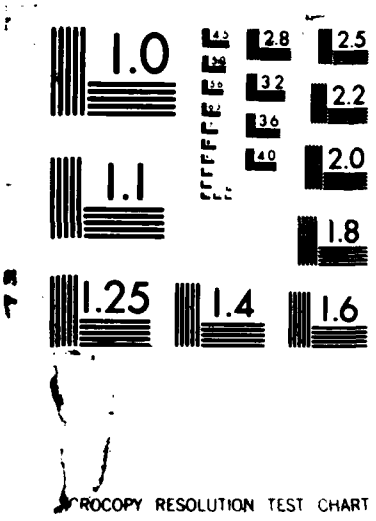
AD-A181 993 THE APPLICABILITY OF STRUCTURED MODELING TO DISCRETE EVENT SIMULATION SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D J PATRICK MAR 87 1/2

UNCLASSIFIED F/G 15/6 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
ROCOPY RESOLUTION TEST CHART

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

THE APPLICABILITY OF STRUCTURED MODELING
TO
DISCRETE EVENT SIMULATION SYSTEMS

by

David James Patrick

March 1987

Thesis Advisor Daniel R. Dolk

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS NONE
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 74
7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)
9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
THE APPLICABILITY OF STRUCTURED MODELING TO DISCRETE EVENT SIMULATION SYSTEMS

12 PERSONAL AUTHOR(S)
Patrick, David, J.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year Month Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 March	124

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Structured modeling (SM) Discrete event simulation
Combat simulation models
Model management systems

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Organizations involved in the development, maintenance and use of combat simulation models have a need for computer-aided model management tools. Structured modeling (SM), a new modeling paradigm developed by Prof. Geoffrion of UCLA, was designed to provide such tools in support of mathematical programming models. This thesis examines the effectiveness of structured modeling when applied to discrete event simulation by attempting to represent an existing combat simulation model using SM. There are three main products of this work.

First, a demonstration of the benefits which accrue from representing a simulation model using SM. Second, a review of the limitations of the structured modeling methodology for discrete event simulation. Third, recommendations for overcoming these problems.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Daniel R. Dolk	22b TELEPHONE (Include Area Code) (408) 646-2260 / 22c OFFICE SYMBOL 54DK

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

The Applicability of Structured Modeling to Discrete Event Simulation Systems

by

David James Patrick
Captain, United States Air Force
B.S., Radford College, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
(Command, Control and Communications)

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

Author: David James Patrick
David James Patrick

Approved by: Daniel R. Dolk
Daniel R. Dolk, Thesis Advisor

John T. Malokas
John T. Malokas, Jr., Second Reader

Michael G. Sovereign
Michael G. Sovereign, Chairman,
Command, Control, and Communications Joint Academic Group

David A. Schrady
David A. Schrady,
Academic Dean

ABSTRACT

Organizations involved in the development, maintenance and use of combat simulation models have a need for computer-aided model management tools. Structured modeling (SM), a new modeling paradigm developed by Prof. Geoffrion of UCLA, was designed to provide such tools in support of mathematical programming models. This thesis examines the effectiveness of structured modeling when applied to discrete event simulation by attempting to represent an existing combat simulation model using SM. There are three main products of this work.

First, a demonstration of the benefits which accrue from representing a simulation model using SM. Second, a review of the limitations of the structured modeling methodology for discrete event simulation. Third, recommendations for overcoming these problems.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Prof. Geoffrion of UCLA, the founder of the Structured Modeling concept, for his help in this thesis effort. Prof. Geoffrion was considerate enough to review some of the problems we encountered using SM and provide some suggestions and guidance. His suggestions were very helpful and are used in sections of this thesis.

I would also like to take this opportunity to thank Prof. Dan Dolk of the Naval Postgraduate School, for his effort on this thesis. Without his help and guidance I would never have encountered Structured Modeling let alone finished the thesis.

I. INTRODUCTION

A. OVERVIEW

Any organization involved in the development, use and maintenance of large software programs has a requirement for a formal computer-aided management system. This is especially true in the area of combat simulation models. The development of a combat simulation model management system for the US Army Training and Doctrine Command Systems Analysis Activity (TRASANA) is currently being investigated by Prof. Daniel R. Dolk of the Naval Postgraduate School. One aspect of this work involves finding a suitable representation for the simulation models which can itself then be stored in the model management system. Choosing an appropriate representation is difficult due to the many requirements which model management imposes. Prof. A. M. Geoffrion of UCLA has developed a framework called structured modeling (SM) [Ref. 1], which seems to provide many of the required capabilities. This thesis will examine the applicability of the structured modeling concept to a combat simulation model environment.

If SM proves to be an adequate tool to provide the logical representation of a combat simulation model, then it would be reasonable to attempt the construction of a combat simulation model management system based on SM. There are many reasons to believe such an implementation would be successful.

1. SM provides a graphical interface for the users to interact with.
2. The resulting logical representation is capable of being represented in a database management system.
3. The precise syntax and rigid structure of SM should facilitate a computer implementation.
4. SM provides for natural language interpretations to assist the user in understanding the model.
5. A complete computer-based environment for SM, as described in Chapter 3 of Geoffrion's monograph, could provide all of these features [Ref. 1: Chapter 3].

The obvious first step is to check the applicability of SM to combat simulation models, to define the pros and cons of such an application, and to document them in a usable form. The pros seem obvious; if SM works then the features of SM mentioned above can be incorporated into the model management system. The focus should then be on identifying and documenting the limitations of SM in this environment so

designers of a model management system can assess the merit of constructing such a system based on SM.

B. PURPOSE OF THE THESIS

1. Methodology

The purpose of this thesis is to investigate the suitability of using the structured modeling concept, a new conceptual framework for modeling proposed by Prof. A. M. Geoffrion of UCLA, to represent and document combat simulation models. The applicability of structured modeling will be tested by taking an existing combat simulation model, the ONEC Model provided by TRASANA, and attempting to represent it in the framework of structured modeling.

No attempt has been made to establish a pass/fail criteria for judging the suitability of SM in the construction of a combat simulation model management system. This thesis only attempts to apply SM to the ONEC Model, provide the resulting SM products, and comment on the strengths and weaknesses of SM in this domain. The assessment of the suitability of SM as a basis for the construction of a model management system is left to the designers of that system.

To be more specific, we did not attempt to represent the current ONEC model as implemented in the ONEC program but rather the original documentation of the model. No attempt was made to review the actual program in operation or the program code. This provided a firm, although outdated, base line from which to work. The fact that the final product represents an outdated abstraction of the program and not the current version of the code does not affect the conclusions reached by this thesis.

Several times, in the process of documenting the simulation model, personnel from TRASANA were requested to review intermediate results and provide comments. This provided a forum to clear up ambiguity in the documentation, provide education on the Army structure inherent in the model, and generate feedback. Due to the difference between the date of the documentation, Oct. 1978 [Ref. 2], and the current state of the actual code eight years later, care was taken to ensure the documentation provided was the only source of information represented in the structured model.

2. Structure of Thesis

The outline of the thesis is as follows. Section 2 describes the ONEC Combat Simulation Model provided by TRASANA. Section 3 provides an introduction to the

concepts of structured modeling. Section 4 presents the structured modeling representation of the ONEC Combat Simulation Model. Section 5 discusses the weaknesses of SM in the discrete event simulation domain. Section 6 summarizes the results of this effort. and contains recommendations for further study. Appendices A, B and C contain the documentation of the ONEC structured model.

3. **External References**

Although this thesis deals with the concept of structured modeling in great detail it is not intended to be a complete reference on the subject. Readers desiring further information are encouraged to review Geoffrion's and other related work directly. Introductory tutorials [Ref. 1,3], detailed examples [Ref. 4,5,6], and comparisons to other modeling approaches [Ref. 7], are all available.

C. SUMMARY

The purpose of this thesis is to test the applicability of the SM concept in the arena of discrete event simulation models. The direct outputs of the thesis are an evaluation of the strengths and weaknesses of such an application and the representation of the Geographical and Movement Representation Sections of the ONEC Model in SM. An indirect by-product of the thesis is a section containing descriptions of problems encountered in applying SM and the chosen solutions. This may prove helpful to anyone attempting to use SM on similar problems.

There is an implicit assumption that if combat simulation models can be represented in SM, understanding of these models will increase from use of the graphical representations of the underlying structures. This may be a valid assumption, and indeed feedback from TRASANA personnel is very positive. However, testing this assumption is beyond our scope and it is left to the reader to determine if this form of abstraction is a useful tool to understand the model.

In fairness to Prof. Geoffrion and SM it should be admitted that SM was not developed specifically to model discrete event simulation systems:

> The main concern of discrete event simulation is mimicking the time dependent behavior of some target system.
>
> Structured Modeling, by contrast, is mainly concerned with representing the pertinent essence of the system itself, and prefers to regard generating the time dependent behavior as a non-modeling task best left to a solver. [Ref. 7: pg. 14]

> . . . .one might ask whether structured modeling can support discrete event simulation.
>
> One possible answer is to prepare a static structured model of the system to be simulated and to compose a (probably procedural) control program that edits the elemental detail tables according to the rules governing the systems dynamic behavior. . . .No such solver has yet been built, and it is not obvious whether the idea is practical. [Ref. 7: pg. 17]

Nevertheless, the objectives of SM are ambitious with respect to its applicability to a wide range of models. It is reasonable therefore to examine how SM works with models for which it was not originally intended. Results of this kind of investigation may either open new areas of application for SM or disclose limitations of the approach or both. Regardless of the outcome, the objective is to provide additional insight into SM and the domains where it most fruitfully may be applied.

It is assumed that the reader is conversant in the fields of elementary directed graph theory, set theory, relational algebra, database theory and software engineering. A basic knowledge of these areas will make the structured modeling concept easier to comprehend and assist the reader in understanding the application of the SM concept to a large software program.

II. AN INTRODUCTION TO THE ONEC COMBAT SIMULATION MODEL

The ONEC Combat Simulation Model is a small part of the Command, Control, Communications, and Combat Effectiveness (FOURCE) Combat Simulation Model. A brief description of the FOURCE model will be provided to show the framework of the ONEC model. Then a more detailed explanation of ONEC will be given.

A. FOURCE

FOURCE is a computerized simulation analysis tool which simulates a limited land war scenario in a standard European environment. Two sides Red, always the attacking force, and Blue , always the defending force, are modeled. This model runs without player interaction and its primary purpose is to examine command and control (C2) issues such as the impact on combat of alternative C2 or intelligence systems.

C2 of the Blue forces is exercised from the division to the battalion level. C2 for the Red forces extends from the army to the division level. The resolution for the C2 is at the level of an individual message, radio, computer terminal, or sensor, and by weapon type, within the various units. This resolution provides a good look at the effectiveness of alternative C2 and intelligence systems in terms of combat information and intelligence flow.

FOURCE deals with issues such as command organization, message generation, communication networks, tactical decision rules, air defense, battlefield environment, unit movement, target acquisition and direct fire engagements. ONEC is a subset of FOURCE which was extracted from the total model and modified so that it could function on its own. It deals with the battlefield environment, unit movement, target acquisition and direct fire engagements. ONEC does not perform the same functions as FOURCE, nor does it operate at the same level of detail or resolution. [Ref. 8]

B. ONEC

The ONEC model was developed by extracting the "Fight the Battle" functional area from the FOURCE model and making the necessary software changes required for this subsection to function on its own. This was done to aid software debug, checkout, authentication, and to assist in data sensitivity analysis. ONEC is much smaller than FOURCE because it lacks most of the functions in the total model. However, it is a

subset of the model and therefore exhibits the same degree of complexity as the overall model. This qualifies ONEC as a suitable subject for testing structured modeling since there is enough complexity to provide a challenging test yet ONEC is still small enough to be manageable.

ONEC has four major functions: geographical description, representation of movement, representation of combat support, and representation of direct-fire engagements. The original intent of this thesis was to model the entire ONEC program using SM. However, this goal was not reached and only the first two functions, geographical description and representation of movement, were modeled. Accordingly, these are the only two functions covered in the following sections.

1. **Geographical Description**

The total battlefield in FOURCE is a rectangle 35km by 138km. It is subdivided into grid cells which measure 1km by 3km. Each grid cell is defined in terms of its location, relief, vegetation, roads in the axial direction and roads in the lateral direction. These features are considered to be consistent over the entire 1 by 3km grid cell. These are the fixed features that describe each grid cell. There are also variable features.

The variable features of each grid cell deal with the locations of items on that grid cell. These items include the various Red and Blue units and smaller components which are also given specific locations. These include command posts, sensors and electronic warfare systems. It is easy to see that these locations are subject to change as the simulation progresses.

2. **Representation of Movement**

Motion in the model is calculated for various entities based on features of those entities and the geography traversed. It is necessary to define the units involved and the attributes of those units before describing the procedural logic used to calculate the motion information. The geographical features were described in the last section. The other entities and their related features will be described in the next sections. Finally the procedural logic which actually combines all of this information to calculate motion will be described.

The units in the game can be divided into two classes. The first class deals with the large organizations such as a battalion or a division. These will be called large units. The second class deals with small items which are associated with the large units. This group can also be divided into two sections. The first is weapons which

are grouped by weapon type. The second section deals with items like the command posts and sensors. This section will be called small units.

The small items, both weapons and small units, are always associated with a large unit for destination, direction and speed information. They are defined by their type, kill range for weapons, and location for small units. The weapons are grouped by weapon type and their location is always considered to be a uniform distribution across the forward section of the host large unit. The large units are defined by their location on a grid cell, size (division, regiment. . .), echelon (1st, 2nd, reserve), type (artillery, maneuver), status (orders, moving, engaged in fight. . .) and associated small items (weapons and small units).

A major difference between FOURCE and ONEC is how each unit receives its orders. Orders give each unit a mission and a destination. In FOURCE the entire process of construction and transmitting the orders is a major facet of the program. In ONEC orders are provided to each unit at the battalion level with no negative impact due to command and control issues. The construction and delivery of these orders is not an item of interest to ONEC. (This information is a byproduct of a meeting with TRASANA personnel.)

All of the important entities involved in the movement representation have now been covered. The remaining information deals with the procedural logic of how these entities relate to derive the required motion information for each unit.

There are two items which must be calculated for each unit that is to be placed in motion: direction and speed. Since the direction of travel is required for the speed calculations it will be described first.

In general, direction is a fairly easy calculation to make. Each unit has a current position and a set of orders which provide a destination. Both the destination and the current location are expressed as a set of (X,Y) coordinate pairs. All travel is considered to be in straight lines without reguard to the roads or the terrain, so the direction calculation is usually just a straight line from the current position to the destination. This is true for the Blue forces and some of the Red forces. It is not true for the Red artillery or 1st echelon Red maneuver battalions. These two cases are handled differently, with the assumption that they will move due west.

Overall the direction calculation is simple. The type of the unit must be taken into account and then one of two direction calculations will be performed. Only three pieces of information, location, destination and unit type, are required. A more

challenging decision must be made to decide if a direction calculation must be performed. [Ref. 2: pg. 5-6 and 5-7]

There is a complicated set of rules to determine if a unit requires a direction calculation. If the unit is not moving and its location does not equal its destination then a direction calculation is required. There are other rules which deal with the type of unit, its echelon, and its mission. In all there are eight pieces of information which may be required to determine if a specific unit requires a direction calculation. [Ref. 2: pg. 5-7]

After the direction calculation is made for a unit, the speed calculations may be performed. Speed calculations are based on the maximum speed possible for a unit and a series of factors which are used to decrease that maximum speed. The maximum speed for any unit is 25 km/hr in friendly territory and 15 km/hr in enemy territory. All other factors will reduce this speed until a final allowed speed is determined for that unit. [Ref. 2: pg. 5-7]

These speed factors take into account the relief and vegetation in a cell, the roads available, the unit's direction of travel, the combat situation, the mission of the unit and the type of the unit. These factors determine the maximum allowed speed for the unit. This speed is then considered in terms of the unit's location with respect to other units, both enemy and friendly, and finally a speed is assigned to the unit. Virtually every aspect of the units and the grid cells are taken into account to make the direction and speed calculations.

As with everything there are exceptions to these rules. Here it is important to note that not all units have direction and speed calculations. In particular Red artillery battalions get their speed from Red maneuver battalions which they are paired with. This function of pairing the units is used as an example in Chapter V and will be explained in detail there.

III. AN INTRODUCTION TO STRUCTURED MODELING

A. BACKGROUND

Since the late 70's Prof. A. M. Geoffrion, of UCLA, has been working on a "general theory of aggregation" for the modeling domain. His work led him to believe that this theory could be realized if models from different disciplines could be represented in a "common format". In the early 80's the development and refinement of this "common format" evolved into what is now called "structured modeling" (SM).

SM has taken on a life of its own independent of the quest for a general theory of aggregation. Accordingly, it has its own goals and objectives, a brief discussion of which follows.

1. Transform Modeling from an Art to a Science

It is generally accepted that there is a large gap between the knowledge domains of model builders and model users, and even between builders of different models which may have to be integrated. This is due to lack of an accepted engineering process by modelers, a problem also experienced in software engineering where the loss of essential information in the documentation process leads to the inability of the users to grasp the detail presented in the model's documentation. SM attempts to reduce these problems by:

1. Providing a framework and formal syntax for models based on five element types and acyclic, attributed graphs.
2. Enforcing a modular design and encouraging the use of stepwise refinement.
3. Easing communications between the builders and users of the models by providing for the presentation of information at various levels of detail which can be tailored to particular audiences.

2. Provide for a Computer-based Modeling Capability

As computer literacy spreads and computing capacity becomes cheaper and more accessible, a trend to more user-developed models will occur. One of the long-term goals of SM is to develop a computer-based modeling capability which will allow a user to conceive an idea and implement the required model as needs dictate. An obvious example of this postulated trend can be seen in the popularity of spreadsheets hosted on personal computers. Users are willing and able to create their own models if given the correct tools and environment.

3. **Integrate Database Management and MS/OR Systems**

Current technology in database management systems provides an extensive array of tools to perform any required data manipulations. However, this technology is very poor in the handling of complex mathematical and logical functions. The MS/OR disciplines works very well with the math and logic functions but are weak in the data manipulation area. With the advent of a generalized computer-based modeling capability, the best features of both of these two fields will be integrated into one system.

4. **Foundation for the Theory of Aggregation**

The search for a general theory of aggregation motivated the effort to find a "common format" for model representation, which then became the concept of SM. The work on SM will eventually lead back to building a general theory of aggregation, with the knowledge that a "common format" does indeed exist.

B. FUNDAMENTALS OF STRUCTURED MODELING

SM is strongly-typed in that all models are composed of basic elements, each of which must be one, and only one of five basic element types: **primitive entity, compound entity, attribute, function,** and **test.** The relationships between the elements in the model are then represented in a framework of acyclic, attributed graphs. These relationship structures are shown at three different levels of detail from the most detailed level to the most abstract: **elemental structure, generic structure** and **modular structure.**

A Structured Model consists of a modular structure coupled with a generic structure and the associated **elemental detail tables** for each of the genera in the generic structure. This provides all of the tools necessary to comprehend the relationships of the basic elements in the original model. It does not however, provide the tools or logic required to run and evaluate the model! The **evaluation** function is responsible for determining the values of the variable attribute, function and test elements and is accomplished by a separate piece of software called the **solver.**

In addition to these basic features, SM offers various other facilities such as: graphical representation of the structures, different ways to tailor the presentation of the modular structure called **views,** and a capability to examine the interrelationships between the elements using a **reachability matrix.** These other capabilities are possible due to a complex indexing system which fully documents the relationships between the elements in the various structures.

Geoffrion explains all of these facets of SM in very precise detail in his monograph. Unfortunately this rigorous explanation is not always easy to understand. In order to provide the reader a more palatable explanation some of the more important aspects of SM will be addressed in considerably less rigorous detail in the following sections. This is done only to aid the reader in understanding SM. Any specific questions not addressed here should be resolved using Geoffrion's works.

In the following Sections examples from the ONEC Structured Model will be used to illustrate various aspects of structured modeling. All of these examples are taken from Appendix A of this thesis. It may be helpful to refer to this appendix to see the overall context from which these examples are drawn.

1. **The Five Element Types**

Although there are five element types in SM it may help to think of these five elements in only two groups: things and information about things. The first two element types, primitive and compound entities, are the actual physical items in the model. They are called entities. The remaining three elements, attributes (and variable attributes), functions and test elements, serve to describe these first two entities. They can be considered as attributes of the things in the model. This perspective may make the following information easier to understand.

a. Primitive Entity

Primitive entities are the basic components of any model and each model must have at least one. The primitive entities form the roots of the generic structure and all other elemental types evolve from or relate to them. They have no mathematical definition and exist only as existential assertions [Ref. 1: pg. 2-2]. This is somewhat confusing because, although they do not have a mathematical definition, the primitive entities like attributes can and often do have values. These values, if required, are shown in the elemental detail tables [Ref. 1: pg. 2-45]. An example of a primitive entity in the ONEC Model would be the SMALL_UNITS. The elemental detail table for the primitive entity SMALL_UNITS would show a distinct identifier for each small unit in this instantiation of the ONEC model. A small section of this elemental detail table is in Figure 3.1. The data has been made up and does not reflect the actual data in the ONEC model.

b. Compound Entity

Compound entities always reference previously defined entities, either primitive entities or other compound entities. They are used to show relationships and

SMALL_UNIT

SMALL_UNIT	Interpretation
cmd post 1	A command post.
radar 1	A search radar.
radar 2	A height finder radar.

Figure 3.1 SMALL_UNIT Elemental Detail Table.

associations between these already defined entities or to define a new entity. They are the counterparts of intersection files in relational database theory. An example of a compound entity in ONEC would be the ASS_UNIT. This shows the relationship that exists between the primitive entities SMALL_UNITS and LARGE_UNITS, reflecting that each large unit may have one or more small units. The elemental detail table for the compound entity SMALL_UNIT would show the identifier for a SMALL_UNIT paired with the identifier for a LARGE_UNIT. A section of this elemental detail table is shown in Figure 3.2.

ASS_UNIT

LARGE_UNIT,	SMALL_UNIT
unit 1	cmd post 1
unit 1	radar 1
unit 2	radar 2

Figure 3.2 ASS_UNIT Elemental Detail Table.

c. Attributes

Attributes are used to associate certain properties and specific values of these properties with certain entities. Attributes can be either fixed or variable. A **fixed attribute** is one where the value will not change during the evaluation of the model. An example would be the attribute LOC_GRID_CELL for the primitive entity GRID_CELL. It should be obvious that the location of the grid cell will not change during the evaluation process. A **variable attribute** is one whose value is expected to change in the evaluation of the model. An example would be the variable attribute

LOC_LARGE_UNIT for the primitive entity LARGE_UNIT. It is clear here that the location of the units in the model is expected to change in the evaluation of the model.

The attribute values, for both the fixed and variable attributes are also shown in the elemental detail tables of the primitive and compound entities which they describe. For example the elemental detail table of the primitive entity SMALL_UNIT would show the values for the attributes LOC_SMALL_UNIT and SMALL_UNIT_TYPE associated with that specific small unit. The structure of this table would look like :

SMALL_UNIT

SMALL_UNIT || LOC_SMALL_UNIT SMALL_UNIT_TYPE

Attributes may only describe primitive or compound entities. There is no restriction on the number of these entities which may be associated with an attribute.

d. Function

A function is a rule for assigning a value. It is a more sophisticated attribute entity in that the values it assigns are conditional and depend on the current values of the other involved entities. The logic and syntax for defining the **generic rule section** of the function entity are spelled out in Reference 9 . Functions may call any of the five element types.

It is important to note that the function entities are just expressions which produce numeric values for the primitive and compound entities. They are not intended to provide the procedural logic inherent in the underlying program. For example, the function element may provide the logic required to calculate a value but it would not provide the logic which would dictate when this calculation should take place. As Geoffrion states:

> A structured model itself provides no means for performing evaluation by applying the rules of function and test elements. This is a task for a problem solver external to the model. [Ref. 1: pg. 2-7]

The problem solver mentioned by Geoffrion is part of the **evaluation phase** and will be described in further detail later in this section.

e. Test

Test elements are function elements with a range of two values: True and False. They are used anywhere a boolean flag might be required. The syntax for the generic rule section of the test entity are the same as those for the function entity.

2. **The Three Structure Formats**

a. Elemental Structure

All models are composed of, act on, generate and are defined in terms of certain elements. Examples of these elements from the ONEC Model would be grid cells, units, locations, orders, missions, speeds and lines of sight. The **elemental structure** is the collection of all these elements and their inter-relationships. Geoffrion defines the elemental structure as ". . .a nonempty, finite, closed, acyclic collection of elements" [Ref. 1: pg. 2-4]. At the elemental structure level every single element is shown along with the information on which elements are associated with it. This information is obviously necessary but at this level of detail not very useful. This is where the **generic structure** and elemental detail tables provide an additional level of abstraction while still retaining access to the original level of detail. No information is lost with this abstraction because all of the elemental information remains in the elemental detail tables of each **genera.**

b. Generic Structure

In the generic structure all of the like elements from the elemental structure are partitioned into one of the five element types described above. Each grouping of like elements is called a **genus.** The total partitioning of the elemental structure results in a mutually exclusive and collectively exhaustive set of genus called **genera.**

In order for elements to be grouped in the same genus they must satisfy the property of **generic similarity.** This means each element in a genus must be associated with elements of the same genera as every other element in that genus. In other words every item in a genus acts on and is acted on by the same genera.

The obvious example of a genus in ONEC would be the grouping of all grid cell elements into the genus GRID_CELL. A less obvious example would be the grouping of a set of calculations resulting in a true or false answer into a test element genus. This is the case for the CALC_DIRECTION test element , where information about each UNIT is considered and a decision is made as to whether a direction calculation is required for that UNIT.

c. Modular Structure

The modular structure is a flexible tool which allows the user to aggregate the genera from the generic structure into groups which are meaningful to the user. The user may divide the **generic structure graph** any way he sees fit as long as the monotone ordering, where genera only reference genera already defined in the graph, remains intact. In other words no forward references are allowed in the structure.

These different modular structures are called **views** and they allow the user to tailor the presentation of a structured model to different audiences. Different views can be used to change the level of detail, or area of emphasis according to the needs of the presentation.

An example from ONEC might be a view which groups everything directly related to a grid cell into a **module** called &GRID. (The "&" signifies a module.) This would greatly simplify a presentation not concerned with the physical layout of the battlefield by suppressing the associated genera GRID_CELL, RELIEF, VEGETATION, ROADS_AXIAL, ROADS_LATERAL and LOC_GRID_CELL into the module &GRID.

3. **Indexing**

Indexes are used to symbolically identify specific elements in a genus, or establish the relationship between specific elements in different genera. They are used in three different places: the **symbolic genus index,** the **generic calling sequence,** and the **generic rule section.** These three areas and a related topic the **index set statement** will be addressed in the following Sections.

a. Symbolic Genus Index

Each genus is composed of a finite set of one or more elements. Each of these elements can be specifically identified by its position in the elemental detail table of that genus. To represent a typical element in a specific genus a ***unique*** lower case alphanumeric index is used. There are three cases to consider.

The **self-indexed genus** is used when the elements in the genus are important in and of themselves. Examples from ONEC shown with their indexes are: WEAPONw, GRID_CELLg and LARGE_UNITu. It is important and meaningful to be able to reference a specific element in each of these genera.

The **externally indexed genus** is used when a genus is related to one or more other genera. A good example from ONEC is the attribute RELIEF. By itself a value for RELIEF is meaningless. Only when it is combined with a specific grid cell does it

begin to have meaning. Therefore, it would be shown as RELIEFg, with the unique index 'g' associating it with a specific grid cell.

The **unindexed genus** is used when there is only one element in a genus. An index is not required because any reference to the genus completely defines the element required. An example from ONEC is the genus IBL. There is only one International Boundary Line in the model.

b. Generic Calling Sequence

Every genus has a calling sequence, composed of genus names, which identifies all other genera which are called by that genus. For example genus A is said to call genus B if genus B shows up in the generic calling sequence of genus A. Graphically this is represented by a directed arc extending from genus B to genus A. The indexes of the genera in that calling sequence allow the identification of specific elements from a specific genus. This use of indices completely defines the cross-references that exist between the elements. It can also be used to build the graphical presentation of the generic structure. An example from ONEC:

```
ROAD_SPEED_FAC(SPEED_FAC_AXIALg, SPEED_FAC_LATERALg, DIRECTIONu)/f/
```

This shows the genera SPEED_FAC_AXIAL, SPEED_FAC_LATERAL and DIRECTION are called by the genus ROAD_SPEED_FAC directly. It also shows, through the use of the indexes, that it is the value of SPEED_FAC_AXIAL and SPEED_FAC_LATERAL for a specific grid cell element 'g' and the DIRECTION for a specific unit element 'u' which are to be used in the calculations.

c. Generic Rule Section

The generic rule section of the function and test elements is an expression which generates a numeric value for an element in a genus. This expression is essentially a formula which acts on specific elements to provide a numeric value. The genus name and associated indices are used to define the specific elements involved in the formula. An example from ONEC:

```
COMBINED_SPEED_FAC_CELLgu = SPEED_FAC_CELLg + ROAD_SPEED_FACu
```

This shows the combined speed factor for a cell is indexed to a specific grid cell 'g' and a specific unit 'u'. The formula used to calculate this value uses the speed factor for cell 'g' and the road speed factor for unit 'u' which is located on cell 'g'. In all of the above cases, the indices 'g' and 'u' refer to a specific grid cell and unit.

d. Index Set Statement

The index set statements do not directly use the indexes but are used to describe the size of the elemental detail table for that genus. If omitted then the resulting data set defaults to the set of all possible combinations of the elements in the involved genera. An example from ONEC:

Select {LARGE_UNIT * SMALL_UNIT}

The '*' operator stands for the natural join operation and means select only data elements from these two data sets which share identical symbolic indices. The resulting data set will be a list of every large unit and all of the small units associated with that unit.

4. **Evaluation and the Solver**

Evaluation is the process of exercising the structured model and computing values for the function, test and variable attribute elements. In a true acyclic structured model this process can be accomplished in a single pass because all of the genera always call genera further up the graph. Evaluation is done by a software package called a solver.

The actual logic which must be built into the solver is unclear and Geoffrion's work is not very informative in this area. As a minimum the solver must accomplish the following functions:

1. Resolve the **symbolic genus indices** as required to identify a specific element from the elemental detail tables.
2. Resolve the indices in the **generic calling sequences** in accordance with the **index replacement options** [Ref. 1: pg. 2-41]. This is required in order to identify a specific group of elements from a genus or the intersection of two or more genera. In ONEC this might be required to find all grid cells which are occupied by red units. Note this would require a subset of the intersection of the genera GRID_CELL and LARGE_UNIT.
3. Evaluate the logic in the generic rule section of the function and test elements.
4. Update the elemental detail tables to reflect the evaluation of the variable attributes and function and test elements.

5. **Elemental Detail Tables**

An understanding of the function performed by the elemental detail tables is essential to understanding the overall process of SM. So far everything discussed deals with the logical representation of a structured model. Special attention has been paid to the aggregation of all elements into the five element types and how these five element types can be placed into a structure which shows the relationships that exist

between them. Very little has been said about the actual data elements which must populate these five element types. This information is contained in the elemental detail tables.

Everything in SM relates directly to the elemental detail tables. The primitive and compound entities provide the keys to the tables. The attributes, functions and test elements provide the values for the tables. The index set statements define the size of the tables. The indices themselves point to specific elements or groups of elements in the tables. The solver manipulates the values in the table in order to evaluate the model and then stores the results of this evaluation in the tables.

In the simplest case a table is built for each genus and the data is inserted. Each table shows the data value and the values of the elements in the generic calling sequence of that genus. This leads to a case where many tables are identical except for the value column. This happens when several entities have the same identical generic calling sequence. The second step in the process is to join all of these nearly identical tables into one table. This is accomplished by establishing a table with all the elements found in the identical generic calling sequences of these genera and then adding a column for each one of the unique values.

Consider the following generic structure statements:

RELIEF(GRID_CELLg) a;
VEGETATION(GRID_CELLg) a;
ROADS_AXIAL(GRID_CELLg) a;
ROADS_LATERAL(GRID_CELLg) a;
LOC_GRID_CELL(GRID_CELLg) a;

Each of these attributes has the identical generic calling sequence i.e. GRID_CELLg. So the resulting elemental detail table would combine all of these values into one table which would be keyed on the value for the grid cell. The resulting table definition would be as follows:

Name Columns

```
GRID_CELL      GRID_CELL || RELIEF, VEG, ROADS_AX, ROADS_LAT, LOCATION
```

This table would have 6 columns, as shown above, and 1610 rows, one for each of the grid cells. This allows all data related to a grid cell and only a grid cell to be grouped in the same table.

This is a gross oversimplification of the elemental detail table structuring process. As the structures get larger and more complicated the process also gets more involved. Reference 1 Section 2.6 has a very thorough explanation of the process in which should be consulted for further detail.

C. SUMMARY OF STRUCTURED MODELING SYNTAX

Geoffrion's monograph on Structured Modeling includes a table which outlines the syntax for each of the five basic element types. This is included in Figure 3.3 for easy reference [Ref. 1: pg.2-34]. To further clarify the syntax a brief explanation of each section in the formats is included in the following paragraphs.

Genus Type	Format of Genus Paragraph
Pri. Entity	GNAME<i> /pe/ <Index Set Statement> Interpretation
Compound Entity	GNAME<i> (Generic Calling Sequence) /ce/ <Index Set Statement> Interpretation
Attribute	GNAME<i> (Generic Calling Sequence) /a or va/ <Index Set Statement> <:Generic Range> Interpret.
Function or Test	GNAME<i> (Generic Calling Sequence) /f or t/ <Index Set Statement> ;Generic Rule Interpretation

Figure 3.3 Structured Modeling Syntax.

1. GNAME

This stands for the genus name. It is the name assigned to a class of elements grouped into a genus. It is a unique, upper case, mnemonically useful character string with no imbedded blanks which always begins with a letter. An example from ONEC is LARGE_UNIT.

2. Symbolic Genus Index

This is optional and used according to the guidelines explained in Section 3a above. When used it is a unique lower case alpha-numeric character string appended to the end of the genus name. It must start with a letter. It is generally refered to as just the index. Example from ONEC is LARGE_UNITu.

3. **Genus Type**

This is required for all of the element types and serves to identify which of the element types is being used.

1. /pe/ Primitive entity
2. /ce/ Compound entity
3. /a or va/ Fixed or variable attribute
4. /f or t/ Function or test element.

4. **Generic Calling Sequence**

This is not used for the primitive entities because they do not call any other genera. It is mandatory for the other four element types. It is a list of genus names and their indicies set off with parentheses. An example from ONEC: (LARGE_UNITu, SMALL_UNITs).

5. **Index Set Statement**

This is optional but if it is omitted then the resulting set is the set of all possible combinations of the genera in the generic calling sequence. If it is included there are three cases.

1. Unindexed genus - Must be a 1.
2. Self-indexed genus - A number defining the maximum size of the genus. May also use relational operators.
3. Externally indexed genus - This requires a complex formula based on relational algebra. It is not easy to put into simple terms so an attempt will not be made. See Reference 10 for a complete treatment of this area.

6. **Generic Range**

This is used only by the attribute elements. It defines the range and type of the attribute values. It is always preceded by at least one space and a colon. Reference 11 contains the syntax for the generic range statement. An example from ONEC, taken from the RELIEF attribute, is: "5Dd, 5Dc, 5Ec, 5Fc". This indicates that only one of these four values is acceptable.

7. **Generic Rule**

This is used for the function and test elements only. It is always preceded by at least one space and a semicolon. Reference 9 contains the syntax and examples for the generic rule section. An example from ONEC, taken from the ROAD_SPEED_FAC function element, is:

```
;ROAD_SPEED_FACgu =
        {[ SPEED_FAC_AXIALg * @abs ( @cos DIRECTIONu )] +
         [ SPEED_FAC_LATERALg * @abs ( @sin DIRECTIONu )]}
```

[@abs (@cos DIRECTIONu) + @abs (@sin DIRECTIONu)]

8. **Interpretation**

This is used in all five of the element types. It is the English language explanation of exactly what the element is doing. There is no syntax and style is a matter of personal taste.

9. **State Diagram**

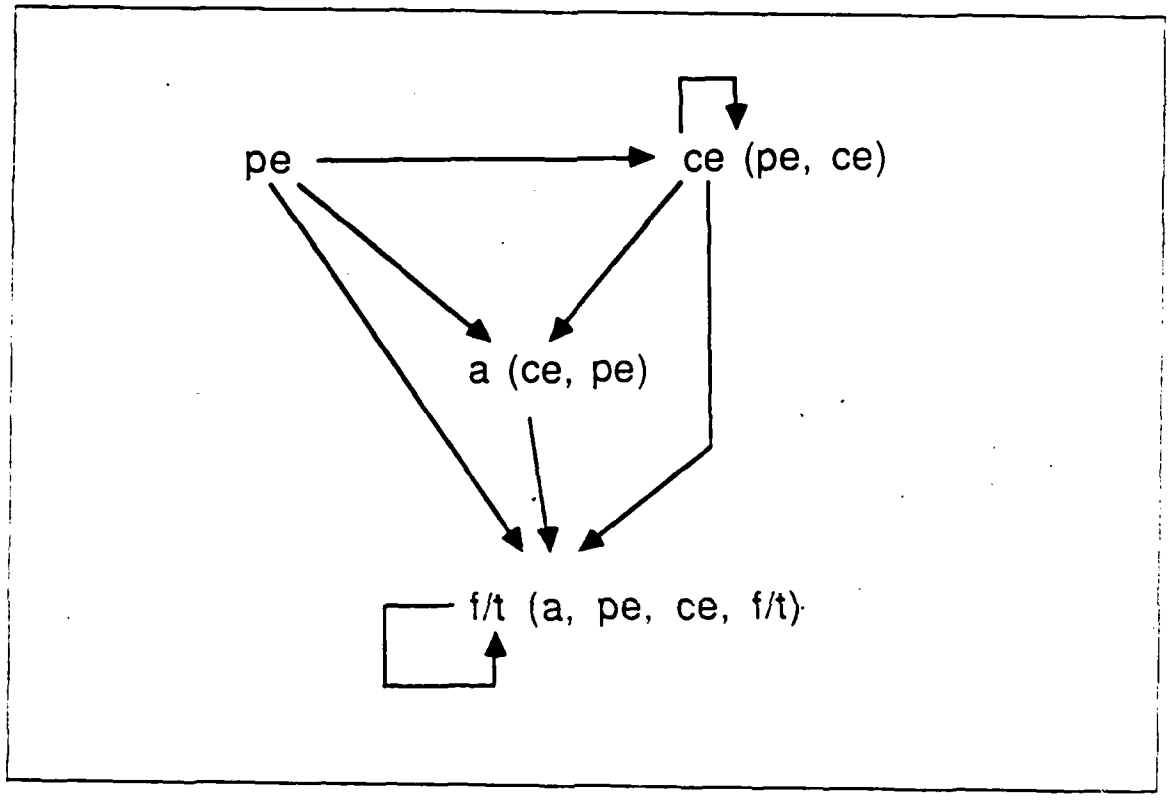


Figure 3.4 Element State Diagram.

There are certain integrity constraints which pertain to the five element types described in the past sections. For example an attribute may not call a function or test element. These relationships are not easy to remember when first dealing with SM. Figure 3.4 is a generic structure of SM which shows the acceptable calling sequences among genera. Figure 3.4 can be read by following the arrows. Any element 'A' which has an arrow pointing to it from element 'B' may call element 'B'. Therefore, an attribute element can call a primitive or compound entity element, but a primitive entity element may not call any elements.

10. Model Schema

A concept used for discussing a model or any part of a model is the **model schema.** A model schema consists of a paragraph for each module and for each genus, ordered and indented to show the modular structure. When it is necessary to focus on a specific instance of a model these schema are supported by populated elemental detail tables. [Ref. 1: Pgs. 2-32 - 2-33] Examples of the ONEC model schema are in Appendix B.

IV. IMPLEMENTATION OF ONEC IN STRUCTURED MODELING

The ONEC Structured Model will be presented in a descriptive manner that points out interesting features of the model, while ignoring the problems for the time being. Although there were many problems encountered in modeling ONEC with SM it is important to view the resultant products without the prejudice brought on by knowing that the model is incomplete. A detailed review of the problems will be provided in Chapter V.

As mentioned earlier, SM is built around five element types organized in structures which document their interrelationships. There are three structure types available each at a different level of detail. These structures from the most detailed to the most abstract are: elemental, generic and modular. It would seem logical to start with the elemental structure. However, in reality the elemental structure is not used much in the model building stage. Rather, it appears in the elemental detail tables which are determined by the generic structure. The generic and modular structures are where most of the model-building occurs so we will start with the generic structure followed by the modular structure and finish with the elemental detail tables.

A. GENERIC STRUCTURE

The creation of the generic structure comprises the primary workload in building a structured model. In this phase most of the modeling decisions are made and fine details are worked out, with the results recorded in the individual genus paragraphs. Accordingly, the generic structure contains virtually all of the essential information about the general model.

Model information is necessary in varying levels of detail, from specifics about individual elements, to the interrelationships between elements within a functional area, to the interrelationships between elements for the entire model. All of this information is available in the genus paragraphs; however, the relationship information is difficult to use and comprehend in this format. To overcome this limitation it is possible to use the relationship information in the genus paragraphs to build a graphical representation of the generic structure in a directed graph. Thus, there are two major aspects of the generic structure: the genus paragraphs and the resultant graphical representation. We will consider the genus paragraph information first.

1. Genus Paragraph Information

Structured modeling is based on things, the primitive and compound entities, information about these things, the attributes, and manipulation of the information which describes the things, the function and test elements. All of these element types are defined by genus paragraphs and provide information about the model. Each element type provides different information.

The primitive entities show the basic units in the model. Everything else either describes the primitive entities, pairs them with other entities, or manipulates information about them. The primitive entity element type is a great aid in understanding a program which does not appear in some other form of software documentation. To demonstrate this point the reader might examine an existing software specification and see how long it takes to determine the key elements in the program which every other element in the program either directly or indirectly depends on. Then turn to Appendix A and see how long it takes to identify the primitive entities in ONEC. A quick glance at the graphical representation of the generic structure immediately reveals the roots of the graph structure as the primitive entities. Experience with the ONEC specification and structured model indicate that SM does indeed help in this regard.

There is other information in the primitive entity genus paragraphs as well. It will show the number of items in that genus, if known, and it provides a plain text explanation which describes the primitive entity. Two examples follow.

IBL /pe/ 1 There is a line called the International Boundary Line. It separates the friendly side of the battlefield from the enemy side.

GRID_CELLg /pe/ Size GRID_CELL = 1610 1610 GRID_CELLS, each measuring 1km X 3km, are placed on a 35Km X 138km Battlefield with their long sides parallel to the long side of the Battlefield.

From these two examples we see there is a single IBL and 1610 grid cells in the ONEC model. There is also an explanation of exactly what an IBL or grid cell is.

Compound entities can also be considered as describing things, but not in the same way that the primitive entities do. The compound entities show the relationships which exist between other entities. In this sense they act like relationships in the entity-relationship database model. There are many cases in a model where it is not the key elements which are of primary interest but rather their interaction. In a structured model a compound entity can be used to show this interaction. An example follows.

LARGE_UNITu /pe/ There are many LARGE UNITS to be considered in this model.

WEAPONw/pe/ There are many Weapons in this model. This approach assumes that weapons are accounted for by groups in weapon types, not as individual units.

WEAPON_LIST(WEAPONw, LARGE_UNITu)/ce/
Select {WEAPON} X {LARGE_UNIT}
Where w covers {LARGE_UNIT}
Each LARGE_UNIT has a list of all WEAPONS associated with that UNIT.

This example shows that there is a relationship between the weapons and the large units and tells what that relationship is, namely, that each large unit has a specific complement of weapons.

The attribute elements provide the modeler the ability to build a wide variety of data types with which to define the entity elements. This capability is very much like the feature of abstract data types which appear in new high order languages like Pascal and Ada. This should help to make the model easier to understand since the modeler can build data types which resemble the real world objects being described. Three examples follow.

LOC_GRID_CELL(GRID_CELLg) /a/ {GRID_CELL} : (0 = < X < 135, 0 = < Y < 135) The location of each grid cell is shown as an ordered pair of (X,Y) coordinate pairs. The first pair represents the NE corner of the unit. The second pair represents the SW corner of the unit.

ROADS_AXIAL(GRID_CELLg) /a/ {GRID_CELL} : "none, primary, secondary, both" Each GRID_CELL has a value for roads in the axial direction.

VEGETATION(GRID_CELLg) /a/ {GRID_CELL} : 0 < = INT < = 10 Each GRID_CELL has a value associated with it that tells the fraction of the cell covered by vegetation.

These examples show three different data types used to describe a grid cell. The LOC_GRID_CELL attribute is an ordered pair of X,Y coordinate pairs. This is a nice alternative to a Fortran implementation which would define four distinct variables for the location information. The ROADS_AXIAL attribute uses character strings to represent information which might normally be encoded with a numeric value. It is obviously much easier to read "none" and understand what is meant than it would be to see a "1" and have to look up what it stood for. The final example, VEGETATION, shows that numeric values are also valid data types. The ability to create a data type suited to the need is a valuable tool in building an understandable model.

There is a great deal of information stored in the attribute genus paragraphs. First, by looking at the generic calling sequence section, it is always clear what entity

or entities, in the case of an attribute which calls a compound entity, the attribute is a property of. Second, there is an indication of how many attribute values are required. This can be found in the index set statement section, where the population of the attribute elements is defined. In each of the above three examples, the index set statement shows one value for each attribute for each of the 1610 grid cells. Third, the exact range of each attribute is shown. The examples show this done by complete enumeration, in the case of the ROADS_AXIAL attribute, and by an algebraic expression in the other two cases. This is much more flexible than being restricted to data types of real, integer, or character string as in Fortran, for example. Finally, there is the plain text explanation of the attribute. This augments the mnemonically meaningful attribute name and provides an excellent vehicle for data documentation.

The function and test elements provide the tools necessary to manipulate entity elements and attribute values. These function elements provide a very strong mathematical modeling capability and are one of the distinctive features of SM. The test elements are identical to the function elements except that they only generate logical, true/false, values.

Geoffrion's early monograph did not provide a syntax for the generic rule section of the function elements, and left the reader with the impression that this section would be implementation dependent [Ref. 1: pg. 2-36]. Accordingly, many of the generic rule sections are done in a pseudo-code like manner. During the course of this thesis, we received a supplement to Geoffrion's monograph detailing a syntax for the generic rule section [Ref. 9]. Geoffrion's recommended syntax leans heavily to a mathematical notation as opposed to a high order language approach. Because the modeling effort was mostly complete by the time that the supplement was received, very few parts of the model reflect this syntax. Two examples of ONEC function genera using this syntax follow.

DIST_RAB_RMB1ER(LOC_LARGE_UNITu1, LOC_LARGE_UNITu2)/f/
Select{LARGE_UNIT} X {LARGE_UNIT}
; @abs {|(Y1u1 + Y2u1) / 2 - (Y2u2 + Y1u2) / 2 |}
The distance between each Red Artillery Battalion (RAB), index u1, and every Reserve Red Maneuver Battalion of the 1st Echelon (RMB1ER), index u2. The distance is only concerned with the north south separation and is measured from the midpoint of each unit.

MOVING_MIN(MIN_DISTu1u2, MOTIONu2)/t/ {MIN_DIST}
; @if(MOTIONu2 = TRUE), true, false) If the RMB1ER unit paired with the RAB unit is moving then MOVING_MIN is true. This calculation is done for each RAB.

The generic rule section of the function and test elements caused many problems in building the model. However, if the syntax is implementation dependent as Geoffrion suggests, then this could be a very powerful tool. If a high order language capability could be embedded in SM to perform this function, then the function elements could accomplish virtually any task required. An example of the pseudo-code approach follows.

```
MISSION_REL_FACTOR(COM_SPEED_FAC_CELLu,        LOC_LARGE_UNITu,
LARGE_UNIT_TYPEu, MISSIONu)/f/{LARGE_UNIT};
 ; If MISSIONu = DELAY
 then
    MISSION_REL_FACTOR = 0.75
 else
    If (MISSIONu = ATTACK) and
       (TYPE = 1st ECHELON DIVISION)
       then
          Select UNITu * GRID_CELLg
          for LOCATIONu intersect LOC_LARGE_UNIT
          SORT on COMBINED SPEED_FAC_CELL ascending
          MISSION_REL_FACTOR = slowest COMBINED SPEED
          FACTOR CELL
       else
          If (MISSIONu = ATTACK) and
             (UNIT_TYPEu = 2nd ECHELON DIVISION)
             then
                Select UNITu * GRID_CELLg for
                LOCATIONu intersect LOC_LARGE_UNIT
                SORT on COMBINED SPEED_FAC_CELL ascending
                MISSION_REL_FACTOR = fastest SPEED_FAC_CELL
             else
                MISSION_REL_FACTOR = 1
```

MISSION_REL_FACTORS seems to apply to only the BLUE UNITS DELAYING or the RED UNITS ATTACKING. It requires a sorted list of the COMBINED SPEED FACTORS CELL for each CELL that the RED UNIT is sitting on. This requires a link between the UNIT LOCATION and the GRID_CELL LOCATION.

This example is fairly complicated but would be an easy task to program in Pascal. It could be , and probably needs to be, broken down into smaller pieces. A compound entity could be developed which showed the units paired with the grid cells that they occupied. This could be a sorted list within each unit based on the speed factors for the cells. This would reduct the amount of code in this function to a much smaller if-then-else statement.

The appropriate syntax for the generic rule section is still open to debate. If a high order language implementation were possible, it would significantly enhance what the modeler could build. But would this be consistent with the Structured Modeling framework? This issue will be discussed at greater length in Chapter V.

2. Graphical Generic Structure

The elements just defined are all interconnected through the generic calling sequence sections of the genus paragraphs. However, it is virtually impossible to grasp the interrelationships which exist between these elements by viewing the genus paragraphs. Fortunately, this information can be used to build a directed graph representation of the element relationships. The graphical representation of the ONEC generic structure is shown in Figure 4.1.

Figure 4.1 contains every genus element developed in our partial ONEC model but it's doubtful that a figure this "busy" would normally be used. The amount of detail in the figure can be adjusted easily through the use of the modular structures; as discussed in the next section.

There is considerable information available in Figure 4.1 , for example a user might be interested in how the terrain of the battlefield was modeled. By examining the GRID_CELL primitive entity (bottom left of figure) it is easy to see that only five factors are taken into account: location, relief, vegetation, and roads in the axial and lateral directions. Should more information be required the user would now know exactly which genus paragraphs to examine.

A user might also be interested in the impact of terrain on the direction of a unit's travel. It is easy to see by examining the DIRECTION function (middle of figure) that the terrain has absolutely no impact on the direction of a units travel (i.e. DIRECTION is not in the terrain's reachability set). A huge mountain or steep drop-off would not force a change of direction in this model. A closer examination would show the user that only four factors are taken into account when calculating the units direction: location, unit type, unit destination and a boolean flag. This process could be continued by tracing all of the related genera until the user was comfortable that he knew all the factors which could affect the calculation of a unit's direction. All of this information is readily available through the generic structure.

The user might wish to pursue the role of terrain in the model. This can be seen from the figure by following the arrows emanating from the GRID_CELL primitive entity (bottom left of figure). It is clear that all of the attributes of GRID_CELL are used in the functions for calculating speed factors. So while terrain does not impact direction of travel it does play a major role in determining how fast a unit may go.

From these examples it is clear that the graphical representation of the generic structure provides the user with a powerful tool for understanding the model and presenting it to others.

B. MODULAR STRUCTURE

The modular structures in SM provide ways for the user to group the genus elements into structures which are meaningful. The concept behind this grouping is the same as the rationale for grouping individual elements into specific genera. The elements are grouped into genera based on generic similarity, allowing the user to consider a more meaningful grouping than the individual elements. A similar process applies for grouping genera into modules. [Ref. 1: pg. 2-5]

By grouping genera into modules the user can create units at a more abstract level than the component parts. The resulting modules can also be grouped into higher level modules. This nesting of genera into modules and modules into larger modules continues until the entire model is represented by a single module. This creates what Geoffrion calls a "hierarchical conceptual structure" [Ref. 1: pg. 2-5], for the model. The modular structure can also be approached from the "top down" and used as a development tool in addition to the "bottom up" approach just shown. This is discussed further in Section 4 B 2.

The module information of a structured model can be viewed in three different ways. The first two are the textual and graphical representation of the modular structure, which is in an ordered tree form. The third; and possibly most interesting, is the graphical representation of the module graph. All three of these representations will be covered.

1. Modular Structure : Text and Graphical

The modular structure can be shown in both a textual and a graphical format. The textual format uses an indented list to represent the preorder traversal of the modular tree. This is best described in Geoffrion's own words.

> What this means in simple terms is that all nodes of the modular structure tree are listed vertically, one to a line, with indentations of each node proportional to the length of its rootpath; the root node listed first, the nodes of each subtree are contiguous and begin with the root of the subtree, and siblings are always listed in their monotone ordering. [Ref. 1: pg. 2-9]

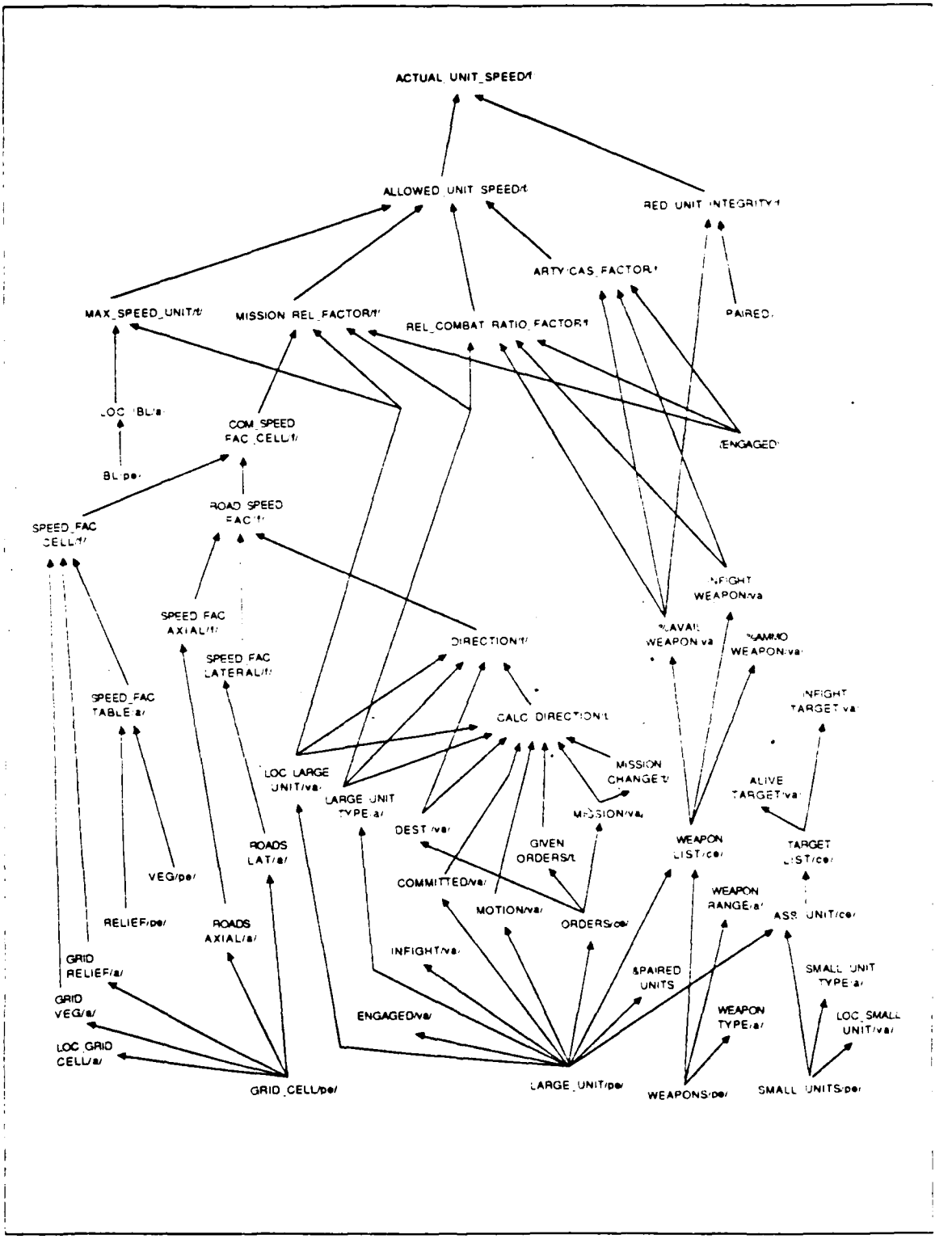


Figure 4.1 Graphical Representation of the Generic Structure.

The translation of the modular structure text into a graphical representation is straightforward and not very interesting. The indented list representation converts into a graphical tree in the obvious manner. An example of a part of the modular structure in text form is shown in Figure 4.2. The corresponding graphical form is shown in Figure 4.3. The full modular structure, both text and graphical, can be reviewed in Appendix B.

```
&MISSION_SPEED
   RED_UNIT_INTEGRITY
   ACTUAL_UNIT_SPEED/f/
   &POSSIBLE_UNIT_SPEED
      ALLOWED_UNIT_SPEED/f/
      MAX_SPEED_UNIT/f/
      MISSION_REL_FACTORS/f/
      REL_COMBAT_RATIO_FACTOR
      ARTY/CAS FACTOR
```

Figure 4.2 Modular Structure Text Presentation.

It is interesting to note that the graphical presentation of the modular structure does not seem to provide any new insight into the model, nor does it seem much easier to use than the text. This was not the case with the textual and graphical presentation of the generic structure; where it seemed that there was a definite difference in viewing the text and the graphics. This leads to the third method of viewing the modular information, the module graph.

2. **Module Graph**

The module graph is just a condensation of the genus graph at any level of detail required by the user. This is a very useful method of presenting module information at varying levels of detail tailored to a specific audience. This is a significant feature of a structured model which should be easy to automate. All of the necessary information is found in the indented modular structure list and the generic calling sequence sections of the genus paragraphs in that listing.

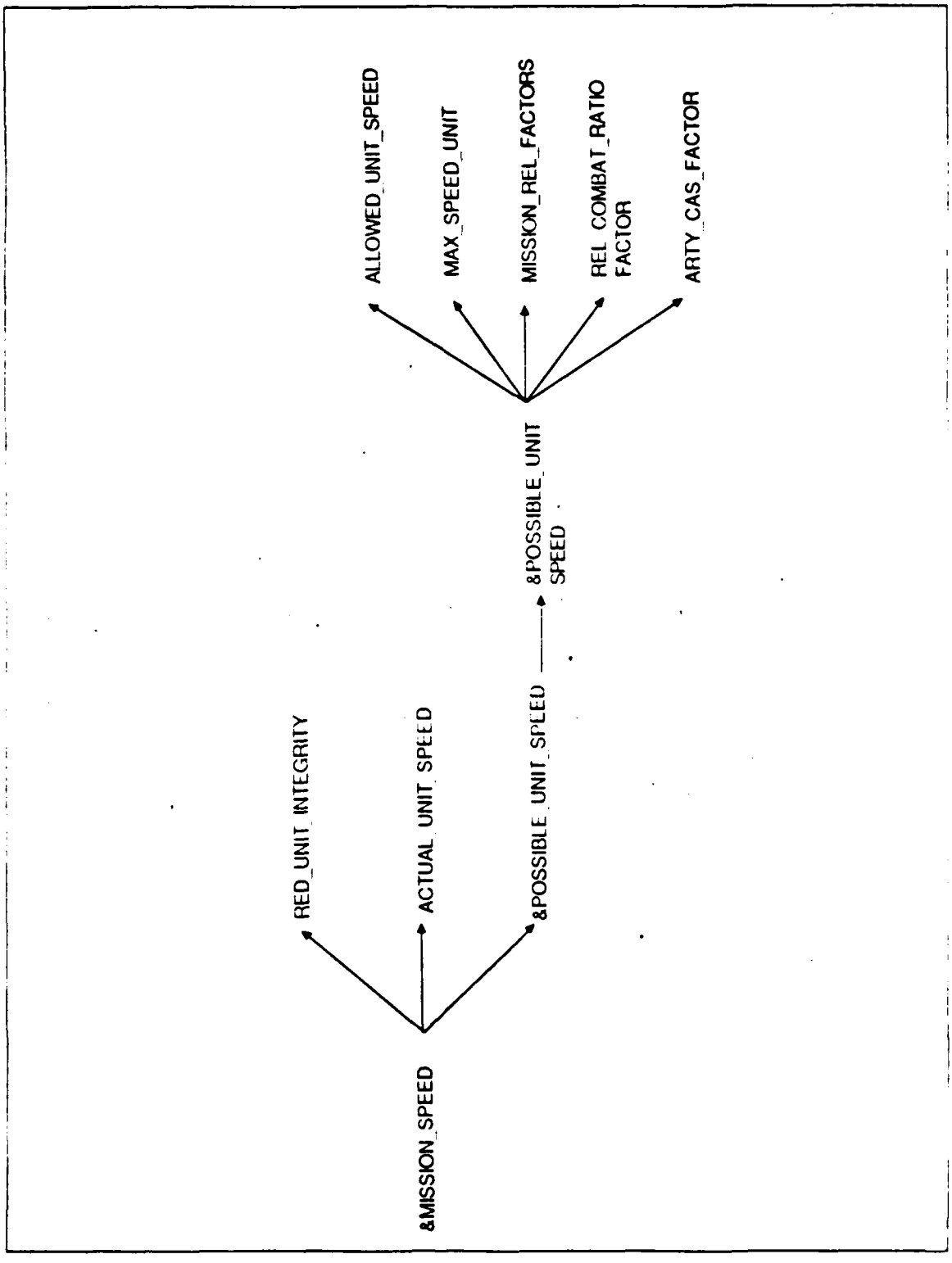


Figure 4.3 Modular Structure Graphical Presentation.

To demonstrate the utility of the modular structure we will present five different views of the ONEC model, each in increasingly greater detail. In the accompanying figures a module is shown with a '&' prefix. Module groups are shown encased by dashed lines with the name of the group set off to the side in a slightly larger font and independent of any arrows.

There are two ways to view the module graphs. The first is as a tool to examine the existing generic structure of an existing model. The second, and a very interesting feature of SM, is to view them as a software engineering tool. One of the goals in SM is ". . .to facilitate top-down model design by stepwise refinement" [Ref. 1: pg. 1-2]. This is handled very nicely through the module graphs. When looking at Figures 4.4 through 4.8 consider them as documentation of a top-down implementation process as well as a method of viewing the model.

Figure 4.4 shows the default modular structure consisting of a single overall module. More modules can be used of course, but in its simplest form, a structured model only requires a generic structure, a modular structure with at least one module, and the elemental detail tables. It can also be considered the very first step in a top-down implementation process.

Figure 4.5 shows a logical division of ONEC into the two major functional areas. This breakdown is exactly what is found in the documentation [Ref. 2: pgs. 5-1 to 5-15]. It should be easy to see that structured modeling can support the software engineering techniques of top-down implementation and stepwise refinement.

Figure 4.6 provides an expansion of the Movement module while leaving the Battlefield module detail hidden. This allows the presentation of the model to focus on the movement issues without the additional detail in other parts of the model.

Figure 4.7 shows a fairly complete system overview without the clutter in the generic graphical presentation shown in Figure 4.1. This level of detail may also correspond to the third or fourth pass through the model design.

The computer graphical presentation should be capable of providing a spectrum of detail ranging from a single module to the entire generic structure. Figure 4.8 shows an expansion which includes some of the actual genus elements. It should also be possible to call up any specific module and examine its graphical structure. A complete listing of each module and the corresponding module graph can be reviewed in Appendix B. A computer implementation should be able to display these modules in any combination required by the user.

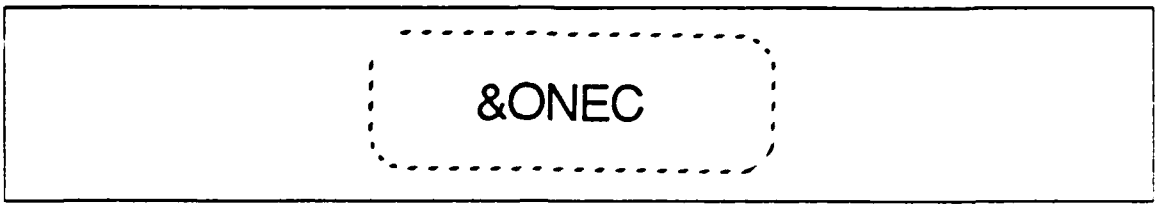


Figure 4.4 Module ONEC.

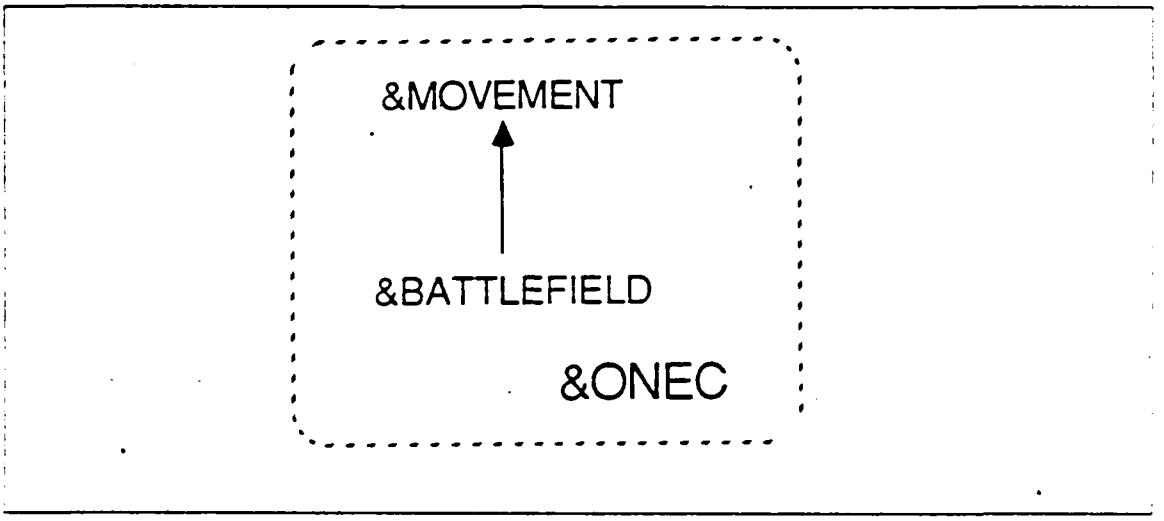


Figure 4.5 Module ONEC Detail.

C. DATABASE REPRESENTATION OF THE GENERIC AND MODULAR STRUCTURES

The last two sections have described the types of information found in the generic and modular structures, but did not address a method for accessing this information. Prof. Dolk, of the Naval Postgraduate School, has developed a way to place the generic and modular structure information into a relational database management system [Ref. 12]. This would allow the user to access the information using a relational query language such as SQL. Some parts of this work are presented here to show the capability of such an implementation. Prof. Dolk's proposed system is based on the Information Resource Dictionary System under consideration by the American National Standards Institute as a prospective Federal Information Processing Standard [Ref. 12: pg.2]. There will not be an attempt to explain these examples in detail as this is covered in the referenced material.

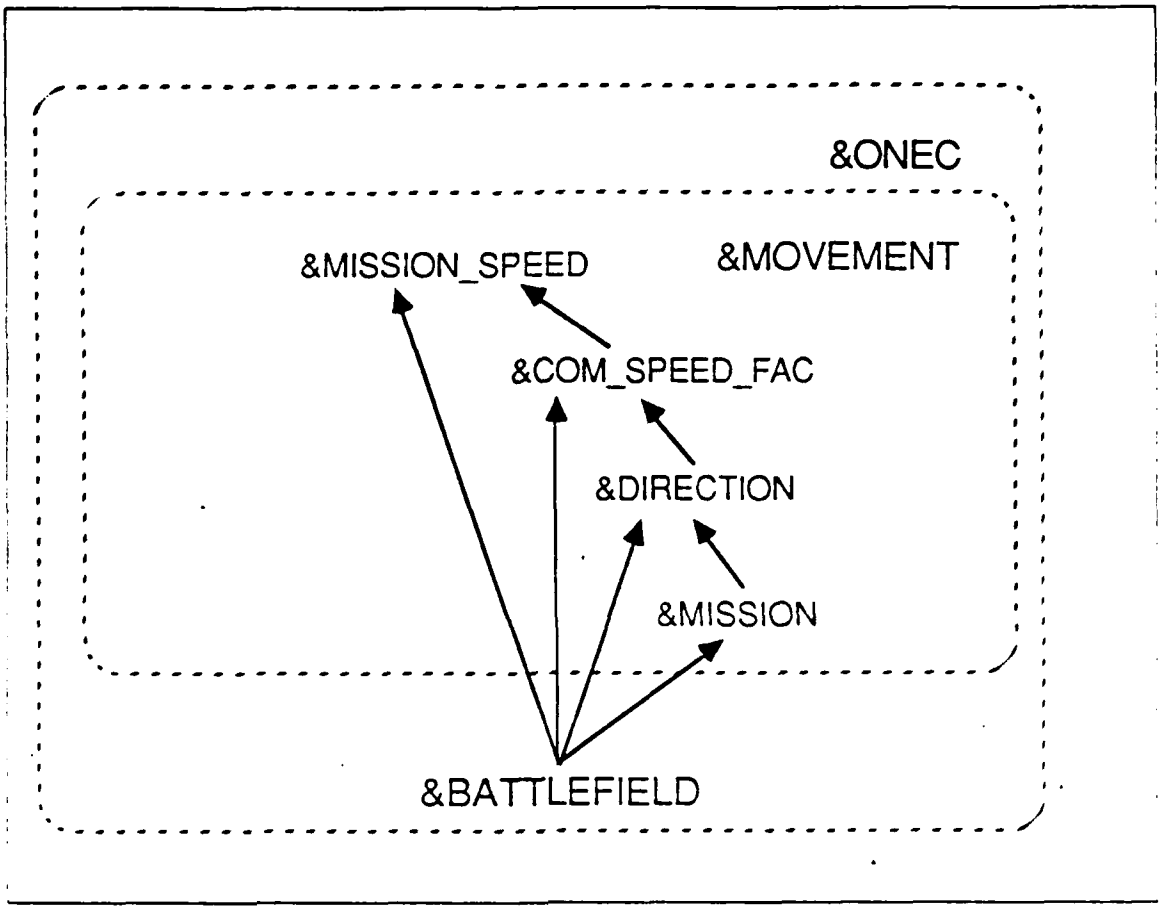


Figure 4.6 &MOVEMENT Expanded.

Prof. Dolk describes exactly how the genus structure information can be placed in a database by creating the required ENTITY-TYPE statements, ENTITY definition statements and INTEGRITY CONSTRAINT statements. He also shows how the modular structure can be defined using the CONTAINS relationship-type statement. A summary of these constructs, taken from Reference 12, is shown in Figure 4.9. Examples of how this would look using the ONEC model information is shown in Figure 4.10.

Once the information has been placed in the database the user has a great deal of flexibility in forming queries concerning both structured modeling and the specific structured model. The examples below, taken from Reference 12 pages 13 and 14, show the types of queries available to the user.

```
SELECT E2NAME FROM CALLS
WHERE E1NAME = 'CE' AND E2TYPE = 'ENT-TYPE'
```

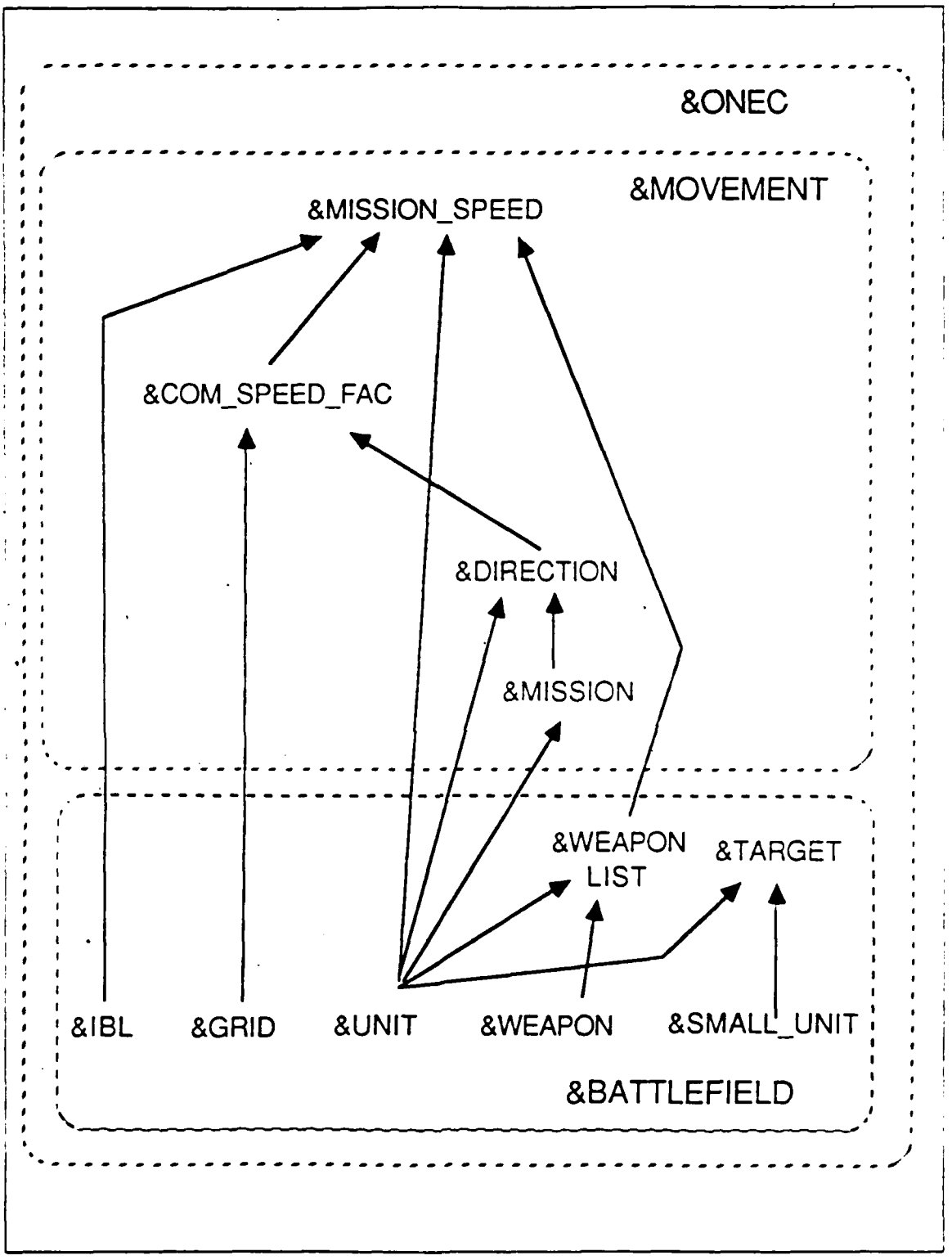


Figure 4.7 &MOVEMENT and &BATTLEFIELD Expanded.

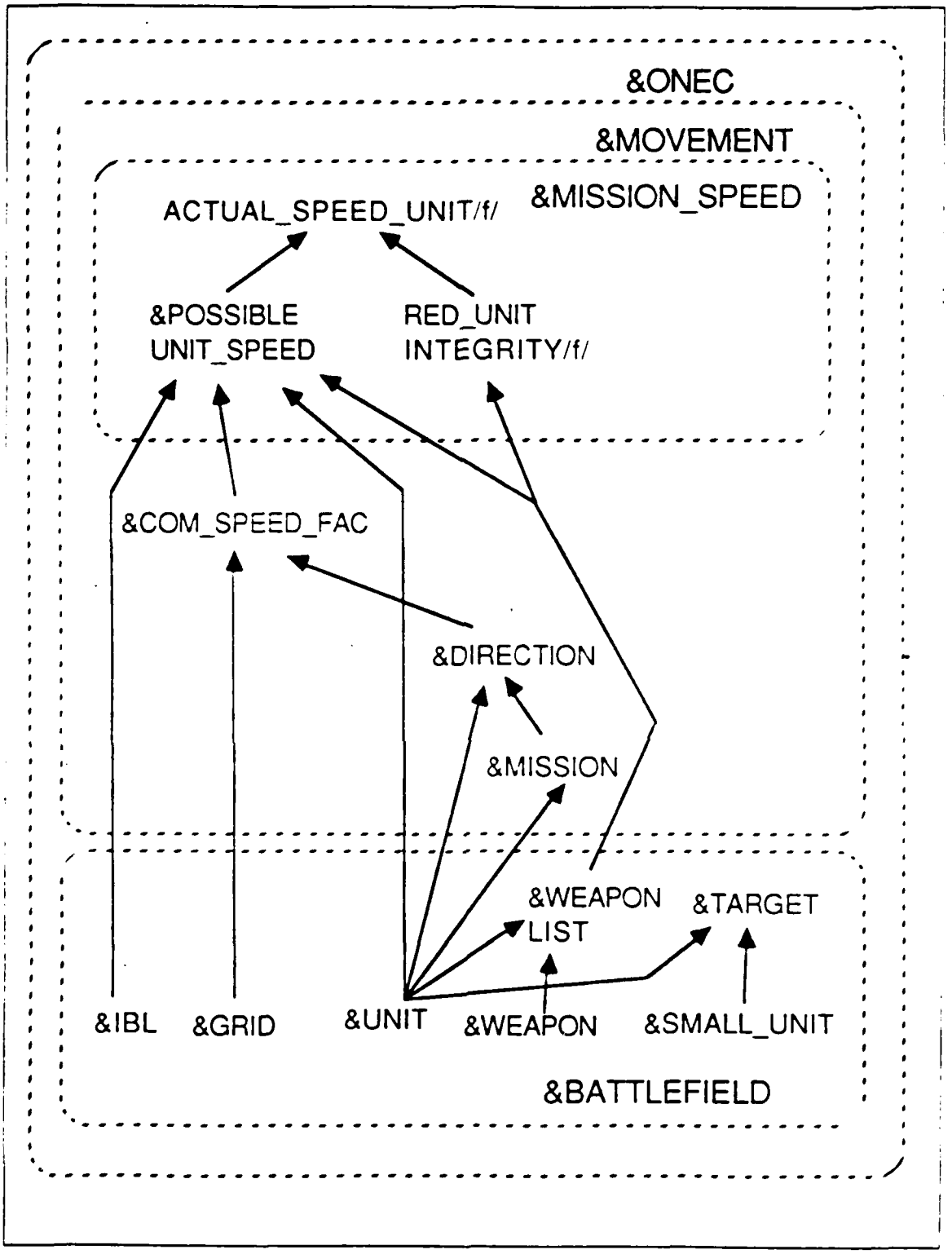


Figure 4.8 &MOVEMENT, &BATTLEFIELD and &MISSION_SPEED Expanded.

Entity-Types

```
ENT_TYPE('pe', 'primitive_entity', ....)
ENT_TYPE('ce', 'compound_entity', ....)
ENT_TYPE('att', 'attribute', ....)
ENT_TYPE('va', 'variable_attribute', ....)
ENT_TYPE('test', 'test_entity', ....)
ENT_TYPE('fcn', 'function_entity', ....)
ENT_TYPE('model', 'model', ....)
```

Entities

```
PE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

CE(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

ATT(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

VA(aname, dname, .... , doc_cat, index,
   index_stmt, gen_range, gen_rule)

TEST(aname, dname, .... , doc_cat, index,
     index_stmt, gen_range, gen_rule)

FCN(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)

MODEL(aname, dname, .... , doc_cat, index,
    index_stmt, gen_range, gen_rule)
```

Integrity Constraints

```
CALLS(ce,pe)        CALLS(va,pe)        CALLS(test,test)
CALLS(att,pe)       CALLS(va,ce)        CALLS(test,fcn)
CALLS(att,ce)       CALLS(test,va)      CALLS(fcn,fcn)
CALLS(test,att)     CALLS(fcn,va)       CALLS(fcn,test)
CALLS(fcn,att)

CONTAINS(module,module)          CONTAINS(module,test)
CONTAINS(module,pe)              CONTAINS(module,fcn)
CONTAINS(module,ce)              CONTAINS(model,module)
```

Figure 4.9 Database Representation of Structured Modeling.

```
                        ENTITIES
MODEL('ONEC', 'ONEC Structured Model',...)
MODULE('&BATTLEFIELD', 'The battlefield..., ...)
MODULE('&IBL', 'The International Boundary Line', ...)
PE('GRID_CELL', '1610 grid cells..., ...)
PE('LARGE_UNIT', 'There are many..., ...)
CE('WEAPON_LIST', 'Each unit has...., ...)
ATT('RELIEF', 'Each grid cell has...', ...)
FCN('MOVING_MIN', @if(MOTIONu2 = T), T, F)

                    GENERIC STRUCTURE

CALLS('RELIEF','ATT', 'GRID_CELL', 'PE')
CALLS('SPEED_FAC_AXIAL','FCN', 'ROADS_AXIAL', 'ATT')
CALLS('WEAPON_LIST','CE', 'WEAPON', 'PE')
CALLS('WEAPON_LIST','CE', 'LARGE_UNIT', 'PE')

                    MODULAR STRUCTURE

CONTAINS('ONEC', 'MODEL', '&MOVEMENT', 'MODULE')
CONTAINS('&MOVEMENT', 'MODULE', '&MISSION', 'MODULE')
CONTAINS('&MOVEMENT', 'MODULE', '&DIRECTION', 'MODULE')
```

Figure 4.10 Database Representation of the ONEC Structured Model.

This would tell the user what SM entity types a compound entity could legally call.

```
SELECT E1NAME, E1TYPE, E2NAME, E2TYPE FROM CALLS
WHERE E1TYPE != 'ENT-TYPE' AND E2TYPE != 'ENT-TYPE'
      AND (E1TYPE, E2TYPE) NOT IN
      (SELECT E1NAME, E2NAME, E2NAME FROM CALLS
      WHERE E1TYPE = 'ENT_TYPE' AND E2TYPE = 'ENT_TYPE')
```

This command would tell the user if the generic structure violated any of the rules of structured modeling.

```
SELECT E1NAME, E1TYPE
FROM CALLS WHERE E2NAME = 'LARGE_UNIT'
```

This would tell the user every genus which called the primitive entity LARGE_UNIT.

```
SELECT E2NAME, E2TYPE
FROM CALLS WHERE E1NAME = 'LOC_LARGE_UNIT'
```

This would tell the user every genus called by LOC_LARGE_UNIT.

The queries available to the user on the structured model and structured modeling are numerous and powerful. Recall, however, that we're dealing with information about the model structure and not the actual data which populates the model. This is the subject of the next section.

D. ELEMENTAL DETAIL TABLES

The first two sections of this chapter presented information about the generic and modular structures. These two structures deal with information about the general model. A distinction must be made between the model schema and a specific instantiation of that schema created when data elements are supplied. The generic and modular structures provide a logical model structure that can be viewed separately from any associated data values. A model instance is comprised of a model structure plus related data values. The elemental detail tables contain these data values.

There are two phases in building the elemental detail tables. The first phase deals with the general model and is the creation of the elemental detail table structure. Creating the structure consists of identifying the table key, the elements required to unambiguously identify a row within the table, and the genus elements which will be the value items in the table. There is a step-by step process for doing this described in Reference 1 on pages 2-46 and 2-52 and covered in Appendix C of this thesis. The second phase is the actual entry of the data in the table structures, thus creating a specific model instance.

The general format of the elemental detail tables is shown in Figure 4.11 . The bold face print shows the required items. The normal print is for explanation only. Some of the more important rules for the table generation are provided below to make understanding these tables easier.

Each table must be named. The name is the genus name of the genus which the table was constructed for. In the case where the tables have been joined, the name of the genus which comes first in the generic structure paragraphs is used. Each table must have an unambiguous key. This is in the section labeled stub columns and includes everything to the left of the double lines. The genus names in the stub columns are those which correspond to the indices in the generic calling sequence of the genus which the table is built for. Finally, each table has a value section, which is everything to the right of the double lines. For the primitive and compound entities there is an optional column which can contain an interpretation of the identifiers. For

attribute, test and function elements the value section will contain the actual values. The number of rows of data in each table is defined by the index set statements of the respective genera.

Since this thesis is only concerned with the development of the structure of the elemental detail tables, and not the loading of the data into these tables, a different format will be used. This format is shown in Figure 4.12. This corresponds to the table name and column heading sections of the table shown in Figure 4.11. The three step process for building the table structure, along with the products of each step , is described in Appendix C.

For illustration several table structures are shown in Figure 4.13. To see how these tables might look when populated with data, the WEAPON and WEAPON_LIST tables are shown loaded with hypothetical data in Figure 4.14 .

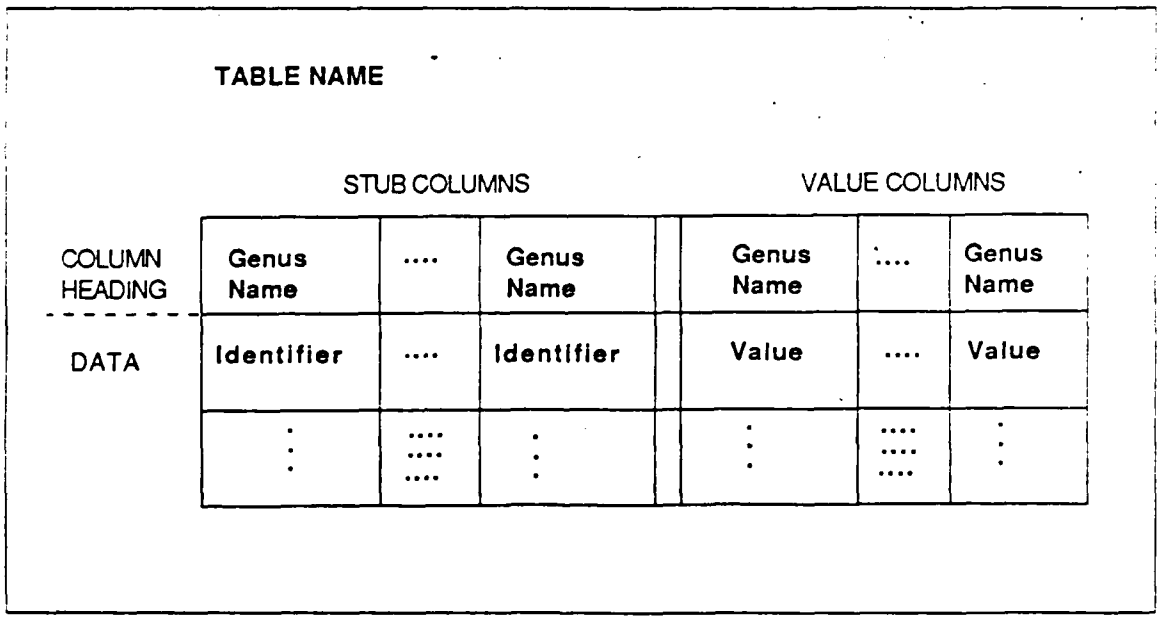


Figure 4.11 Elemental Detail Table Format.

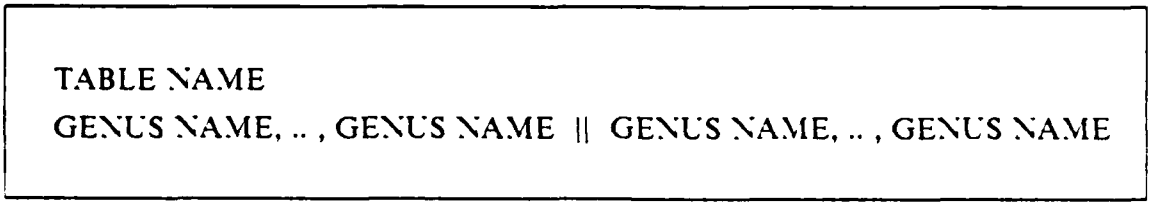


Figure 4.12 Elemental Detail Table Structure Format.

```
LARGE_UNIT
LARGE_UNIT    ||    Interp, LOC_LARGE_UNIT,  LARGE_UNIT_TYPE,
                    COMMITTED, MOTION, ENGAGED, INFIGHT, ORDERS,
                    DESTINATION,     MISSION,    MISSION_CHANGE,
                    CALC_DIRECTION,  DIRECTION,  MAX_SPEED_UNIT,
                    DIST_RAB_RMB1ER,     MIN_DIST,     MOVING_MIN,
                    MISSION_REL_FACTOR,       ALLOWED_UNIT_SPEED,
                    ACT_SPEED_UNIT, GIVEN_ORDERS
WEAPON
WEAPON        ||    Interp, WEAPON_TYPE, WEAPON_RANGE

WEAPON_LIST
WEAPON, LARGE_UNIT  || %AVAIL_WEAPON, %AMMO_WEAPON, INFIGHT_WEAPON
```

Figure 4.13 Sample Elemental Detail Table Structures.

```
WEAPON

WEAPON  ||     WEAPON,      WEAPON
               TYPE         RANGE
tank1   ||     m1           3000
tank2   ||     m48          1800
aac1    ||     redeye       5000


WEAPON_LIST

WEAPON,   LARGE_UNIT       ||  %AVAIL,      %AMMO,       INFIGHT
                               WEAPON       WEAPON       WEAPON
tank1     unit1            ||  90           50           true
tank2     unit1            ||  20           10           true
aac1      unit1            ||  100          100          false
tank1     unit2            ||  100          100          false
tank2     unit2            ||  40           10           true
```

Figure 4.14 Sample Loaded Elemental Detail Tables.

V. PROBLEMS ENCOUNTERED WITH STRUCTURED MODELING

This section deals with some of the problems encountered in the application of Structured Modeling to discrete event simulation. These problems fall into three major categories. The first class addresses areas of discrete event simulation which are in direct violation of the basic concepts of structured modeling and are therefore considered serious major obstacles. The second category concerns areas of discrete event simulation which do not seem to lend themselves conveniently to SM and where stopgap solutions were not easily found. The third class consists of general problems we were unable to model along with proposed solutions, where possible, to the problems.

One problem which appears throughout this thesis is a lack of understanding of the SM process and tools. This shows up in areas where SM tools are incorrectly used or in some cases not used at all. This lack of understanding and ability to use the SM tools has had a profound impact on this thesis.

This problem of comprehension is due in part to the immaturity of the SM concept which manifests itself in several ways:

1. The lack of available documentation in a useable format.
2. The lack of complicated examples which could be copied and studied.
3. The lack of a working SM system which could be experimented with to gain an understanding of the SM process.

Geoffrion is certainly aware of these problems and comments on them in his monograph.

> The presentation of material in this chapter is designed more for completeness and reference purposes than for prospective practitioners of the structured modeling approach. A much shorter, example-based exposition is necessary for the latter group. To them structured modeling will be a new language supported by software; most people assimilate new languages more easily by imitation based on examples than by being lectured on grammar and vocabulary. [Ref. 1: Pg. 2-1]

Working with SM in its current state of evolution must be similar to the tasks faced by programmers in the early 50s. Every time they came upon the need for a data structure, search routine or sorting algorithm they had to invent it; whereas today these are readily available in any introductory text book. SM is in the same state. The tools

are available in SM to build the required model structures but may be beyond the scope of the novice modeler. This will become obvious in the section on modeling hierarchies.

SM is a powerful, but complex, modeling tool which requires a very sophisticated modeler to take full advantage of. It is important to distinguish between problems inherent in the SM approach and those resulting from a lack of modeling sophistication. The distinction is not always clear but we will try to distinguish whenever possible in the following discussion.

A. CRITICAL PROBLEMS

One of the original objectives of this thesis was to examine the impacts of incorporating time into a structured model. Due to problems encountered in trying to build just the static version of the model, this goal was never reached. We were unable to adequately consider the role of time, however, it seems to be characteristic of discrete event simulation models that they are cyclical by nature with respect to time.

1. Cyclical Aspects of a Simulation Model

A classic example, in combat simulation, is the conflict between two units where the attrition factor is based on the power of the units. The original conflict is based on the starting power of the units but as the fight progresses, this unit power value must be adjusted to reflect the results of the fight. The attrition factor must also be adjusted to reflect these changes as the fight continues. This cycling is in direct violation of SM Proposition 2 that Genus graphs always be acyclic [Ref. 1: pg. 2-13]. This unit conflict example comes from a section of the ONEC simulation which we did not reach in our modeling effort, so, we'll consider an implemented example instead.

The example we will use deals with the issues involved in calculating a direction of travel for a unit. Figure 5.1 shows the logic and information required to decide if a direction calculation is required and if so, how it should be done. This is not an accurate representation and serves to illustrate a point only.

The logic is that if a UNIT has ORDERS and it's LOCATION does not equal its DESTINATION and is not in MOTION(at time t), then a DIRECTION should be calculated. After the DIRECTION is calculated the UNIT is placed in MOTION(at time t + 1). At the next pass through the logic the MOTION flag must be set to true and will not change again until the UNIT reaches it's DESTINATION, and the MOTION flag will be reset to false. There seems to be a cycle in these calculations

and we could see no way to model this section without introducing a cycle into the model. Our view was that somehow the model had to loop back on itself to reset the MOTION flag based on the fact that the unit had been placed in motion. We show this in Figure 5.1 as a feedback loop from the module &PUT_UNIT_IN_MOTION to the attribute MOTION. This is not a legal structured model as the attribute MOTION cannot legally call anything other than an entity type genus. It is just shown in this manner to demonstrate that somehow the motion flag would have to be reset.

We posed this issue to Geoffrion, in an informal correspondence, and he was considerate enough to respond and provide a schema which modeled this situation without requiring a cycle. His proposed schema is shown in Figure 5.2.

Geoffrion was able to remove the perceived cycle by removing the MOTION flag while at the same time retaining access to the motion information. He also removed the CALC_DIRECTION flag and the DIRECTION function. His proposed implementation to capture the direction and motion information is shown below.

DIR(LOCt, LOCt+1) /f/ Filter (2 < = t < -1) {T}; LOCt+1 - LOCt

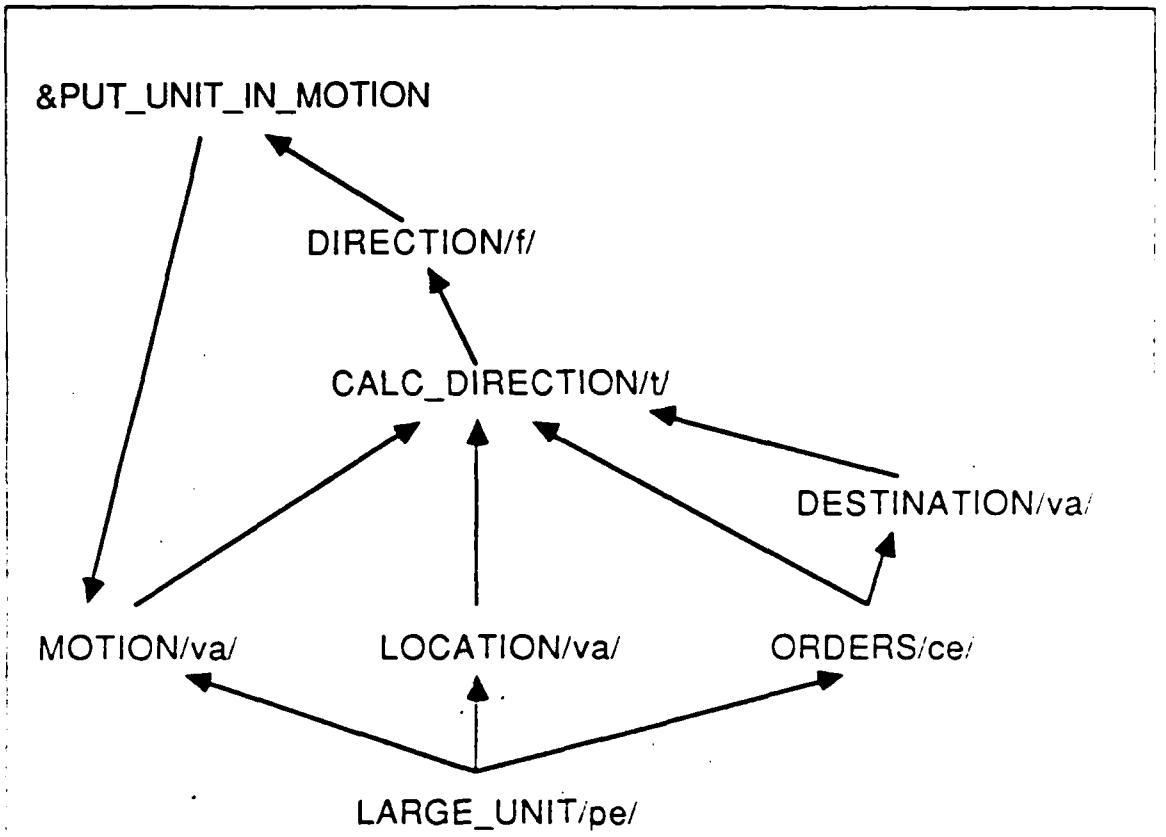
DIR_INIT(LOC2. INITLOC) /f/ ; LOC2 - INITLOC

MOTION(DIRt)/t/ {DIR} ; DIRt < > 0

MOTION_INIT(DIR_INIT) /t/ ; DIR_INIT < > 0

Now that each piece of information is available without a cycle it would also be possible to build the CALC_DIRECTION flag, used later in the model, and the implementations would, from a black box perspective, be functionally identical except for the loop in our structure.

Geoffrion was able to remove this instance of a cycle with an easily understandable piece of modeling. It is possible that he could do the same with other cyclical aspects of the ONEC model. This casts doubts on our assertion that the cyclical aspects of a simulation model would present a "showstopper". We must now consider it a distinct possibility that a ONEC structured model could be constructed without cycles which would ". . .hang together as a static snapshot" (Geoffrion's words). We still present this issue as a critical problem because, in our minds, it is the key technical stumbling block which must be addressed before blessing SM as a tool for discrete event simulation models.



LARGE_UNITu /pe/

LOC_LARGE_UNIT(LARGE_UNITu) /a/ {LARGE_UNIT}

ORDERS(UNITu) /ce/ {UNIT}

DESTINATION(ORDERSu) /va/ {ORDERS}

MOTION(LARGE_UNITu, &PUT_UNIT_IN_MOTION) /va/ {LARGE_UNIT}

CALC_DIRECTION(DESTINATIONu, ORDERSu, LOC_LARGE_UNITu, MOTIONu)/t/ {LARGE_UNIT}
if UNIT has ORDERS and (NOT MOVING) and (DESTINATION < > LOCATION)
then CALC_DIRECTION = true.

DIRECTION(LOC_LARGE_UNITu, DESTINATIONu, CALC_DIRECTION)/f/{LARGE_UNIT}
if CALC_DIRECTION then DIRECTION = (Equations 5-1/2/3/4/5, Pg.5-6)

Figure 5.1 Cycles in Direction Calculations.

It seems that the SM tools can represent and describe the current states of a model very accurately. However, the tools required to model the state transitions do not seem present. The ability to model the dynamic aspects of the simulation programs is a major prerequisite and one which we could not satisfy.

```
Tt /pe/ TIME
UNIT /pe/
ORDERS (UNIT, Tt) /ce/ {T}
DEST(UNIT, ORDERSt) /a/ {T}
INIT_LOC(UNIT, T < 1 >) /a/
LOC (UNIT, INIT_LOC, DEST < 1:t-1 >) /f/ Filter(t > = 2) {T} ;
```

Figure 5.2 Geoffrion's Proposed Schema.

B. MAJOR PROBLEMS

There are two problems discussed in this section and they both deal with the representation of logic in a structured model. The first question deals with the role of logic in a structured model and focuses on the relationship between the solver and the structured model. The second question deals with the tools available in SM to represent the logic of the model.

1. Role of Logic in Structured Modeling

At first we were confused by the apparent division of program logic between the structured model and the solver. After a review of Geoffrion's work and informal correspondence with him on this subject, we have come to the following concept for discrete event simulation models. This concept may not hold true for structured models and solvers used in other modeling domains.

The entire set of logic for a program must be coded into the structured model. The tools available for coding the logic of a program into the model are the generic calling sequences, the index set statements and the generic rules. The solver acts as a kind of super interpreter which takes each genus paragraph, in the order established by a topological sort of the genus graph, and executes the logic in these paragraphs. The

required data for the execution of this logic are found in the elemental detail tables and the results of each step are returned there for use by the genus paragraph in the evaluation process. The evaluation of a structured model only requires one pass through the model. At the end of this pass all of the variable attributes, function and test elements will have values and the model will be fully evaluated.

For a simulation model this process will be slightly different. In accordance with the above concept, the evaluation of the simulation structured model will only represent a single snapshot in time. As a rule, it is not a single snapshot in time that is important, but rather the cumulative effects of multiple time segments. So, the solver would have to execute the model repeatedly, saving the results, until a preset condition had been reached; perhaps a specified number of passes through the model.

This extension of the role of the solver is directly related to how time is implemented in the model and warrants a closer examination. The role of time in a structured model has not been fully examined. One proposed implementation is to create a primitive entity TIME whose elements are each instant in time to be considered by the model. The TIME primitive entity would then be included in the generic calling sequence of every dynamic entity in the model. [Ref. 1: pg. 2-91] An example from ONEC follows.

TIMEt/pe/ There is a list of time instants.

UNITSu/pe/ There is a list of units.

LOC_UNIT (UNITu, TIMEt)/va/ {UNIT} X {TIME} The unit locations.

UNIT_TYPE(UNITu)/a/ {UNIT} The unit type.

Notice how time shows up in the location attribute but not in the type attribute. Only the dynamic aspects of the model would be related to time. It is interesting to examine the impact of this on the elemental detail tables and the solver.

The elemental detail table structure for the above example would be composed of two tables due to the differences in the generic calling sequences. These structures would be as follows.

UNIT_TYPE

UNIT || UNIT_TYPE

LOC_UNIT

UNIT, TIME || LOC_UNIT

The structures are interesting only in the fact that the dynamic and static aspects of the program have been segregated. A much more interesting point is to look at the size of the dynamic tables and the interaction between these tables and the solver.

Notice in the LOC_UNIT index set statement the use of the cartesian product with UNIT and TIME. This will generate a data set where every unit is paired with every time instant. This can be thought of as a three dimensional array with time as the third dimension.

The solver, in its single pass, would evaluate one time slice of the model. Thus, in the first pass every row in the tables indexed to T = 1 would be filled with the variable attribute and function values. The remaining rows would remain empty until the solver completed the pass for that time slice. After the solver has completed its required number of passes the elemental detail tables will be filled up to the row which corresponds to the number of time segments executed. If all of the time segments in the TIME primitive entity were executed then the elemental detail tables will be full.

For a model with a large number of units and/or a large number of time slices this data set will become quite large. The resulting size may be an unacceptable limitation of this approach. Because all of this data is not required, either for analysis or for the execution of the next evaluation pass, it may be worth looking at another option.

A second option is to just save the data of interest and that data required to execute the next pass through the model. Assuming that all of the program logic must reside in the structured model, this would require an extension to SM, probably in the index set statement syntax, to direct the correct sizing of the elemental detail tables and instruct the solver where to read and write the data.

This is considered a major problem because although SM can handle this issue the solution might not be useful due to the size of the data structures required to implement it. The alternative proposed seems workable but it requires a change to the SM syntax and therefore an extensive study in order to implement.

2. **Programming Logic into a Structured Model**

The last section clearly defined the requirement that **all** of a program's logic must be coded into the structured model. The tools available to code this logic were given as the generic calling sequences, the index set statements and the generic rules. The generic calling sequence performs the dual functions of representing the generic structure of the model and identifying specific elements or sets of elements in the genera. The index set statements are used to define the population of a genus. It shows explicitly which elements from each genera are to be brought into the newly formed genus. The generic rules are used to manipulate the values in the model to

produce new values. It is in these rules that the majority of the program logic is placed.

Geoffrion has defined a grammar for the index set statements [Ref. 10], a syntax for the generic rules [Ref. 9], and a syntax for the generic calling sequence [Ref. 1: Pgs. 2-41 - 2-44]. The tools he has provided for these sections are very powerful, incredibly complicated, and the source of the majority of our problems in this attempt at building a structured model.

The syntax for the index set statements and generic rules seem tailored to mathematical models and for a modeler with a strong mathematical background. It is possible, even probable, that these tools are adequate to construct any structure required in the ONEC model; however, they are inappropriate for use by a "programmer" attempting to model a combat simulation program. It is difficult to tell which part of this inappropriateness is the result of the wrong tool for the wrong job and which part is to be laid at the feet of the programmer. Perhaps an example will help.

a. Example of Modeling Problems

An easy way to demonstrate the difficulties faced in the application of SM to the ONEC program is to step through a section of the modeling process. A section of the ONEC documentation dealing with the pairing of the Red artillery battalions and the Red maneuver battalions was chosen because it is a small easily understood section of the model, yet it was complicated enough so that we were never able to completely model it. The section chosen is only one paragraph long; so, it is repeated here for easy reference.

> (5) RED artillery battalions are assumed to move in response to the advance of RED maneuver units. This effect is represented by assigning to each artillery battalion the speed of a selected maneuver battalion. In most cases, the selected unit is the reserve battalion of a first echelon regiment which is nearest in the y (north-south) coordinate to the given artillery battalion. If this battalion is stopped, the most advanced battalion that is either in the first-echelon or has been passed through by another battalion but still has a mission to attack in this regiment is selected and its speed is assigned to the artillery battalion. If no maneuver battalions fit the above criteria or if the RED artillery has advanced to within KBMAXR* (3000) meters of a BLUE maneuver unit, the speed of the artillery battalion is set to zero. [Ref. 2: pg. 5-14]

This short program section can be broken into several function genera which will accomplish the required tasks. We have broken the creation of these function genera into a three step process; which will be used to step through the

modeling effort. The first step is to decide on the genera and indices required in the generic calling sequence section. This provides the function the access necessary to manipulate the data elements. The second step is to define the index set statement which defines the size and population of the resulting elemental detail tables. The third step will be the coding of the generic rule section of the function and test genera. This is the actual logic of the program.

The first task in this program is to calculate the north south distance between each Red Artillery Battalion (RAB) and every Red Maneuver Battalion of the 1st Echelon (RMB1ER). For the purposes of this example we will assume that there are five RAB and five RMB1ER.

Step 1: A technique must be devised to provide access to two sets of elements, RAB and RMB1ER, in the same genus, LARGE_UNIT. We considered having two compound entities, RAB and RMB1ER, and having the function entity call them. However, we decided on a simpler approach of introducing two indices to the LARGE_UNIT genus: u1 for RAB and u2 for RMB1ER. This is done by using the attribute LOC_LARGE_UNIT twice in the generic calling sequence; each time with a different index. This is consistent with Geoffrion's work in Reference 4 pg. 8 and Reference 1 pg. 2-94.

Step 2: The elemental detail table must be sized to hold a value for each possible RAB, RMB1ER pairing. This would require a table that was 25 X 3. The three columns are for RAB, RMB1ER and the function value. The 25 rows are for each possible combination of the five RAB and the five RMB1ER.

Step 3: Build the function rule. This is straight forward because this is a simple mathematical problem which is very easy to do with the SM syntax.

Resulting Genus Paragraph:

DIST_RAB_RMB1ER(LOC_LARGE_UNITu1, LOC_LARGE_UNITu2)/f/
Select{LARGE_UNIT} X {LARGE_UNIT}
; @abs {[(Y1u1 + Y2u1) / 2 - (Y2u2 + Y1u2) / 2]}
The distance between each Red Artillery Battalion (RAB), index u1, and every Reserve Red Maneuver Battalion of the 1st Echelon (RMB1ER), index u2. The distance is only concerned with the north south separation and is measured from the midpoint of each unit.

Comments: The generic calling sequence and the generic rule section look good. However, it is not clear who, the modeler or the solver, must keep track of the indices. The index set statement looks weak. We know exactly what the resulting data set must look like, but we cannot express it. In particular, there is nothing explaining the selection criteria.

The second task in this program is to examine the just created 25 X 3 data set produced by DIST_RAB_RMB1ER and select one pair for each RAB unit which has the shortest distance between the units. This should generate a 5 X 2 data set. The two columns should be RAB and RMB1ER and the 5 rows would be for the 5 RMB1ERs associated with each RAB.

Step 1: The generic calling sequence is just the function DIST_RAB_RMB1ER and the indices u1 and u2 from that function.

Step 2: There seems to be no way to build a 5 X 2 data set. The only way to do this here would be to use a compound entity; which is illegal because a compound entity cannot call a function. Since we are dealing with a function the smallest data set possible would be 5 X 3. The columns would be RAB, RMB1ER and the function value.

Step 3: A key question here is what should the function value be? It is not the distance information which is important, but rather the unit pairs of the two units which share that minimum distance. Since a function must generate a numeric value, how should this be done? We elected to have the function return the index value of the RMB1ER closest to the RAB.

Resulting Genus Paragraph:

MIN_DIST(DIST_RAB_RMB1ERu1u2)/f/ Select{DIST_RAB_RMB1ER} : @and [@min (DIST_RAB_RMB1ERu1.), ord(u2)] This should generate a 5 X 3 data set. The 3 columns would be the RAB, RMB1ER and the specific index of the RMB1ER in the LARGE_UNIT elemental detail table. The 5 rows would be for the 5 RAB.

Comments: The syntax is probably incorrect in the generic rule section; although it should be possible to do what is required. It is a minor inconvenience to have to generate a numeric value when all that is required is the pairing of the units. Again the index set statement lacks any significant information. All it shows is that the resulting data set will be a subset of DIST_RAB_RMB1ER. There is no information on how this subset is chosen. It is also not clear that it is legal to use the function entity in the index set statement. If we are required to use the genus LARGE_UNIT then this index set statement will provide even less information. This index set statement might look like: **Select {LARGE_UNIT} Covering u1.**

The third task in this program is to examine these five RAB, RMB1ER pairs and see which the RMB1ER units are moving. This requires a test genus.

Step 1: It is clear that the 5 X 3 data set from MOVING_MIN and the MOTION attribute for LARGE_UNIT are both required for this function, but it is not clear what the indices should be. For the RAB it is obvious that the index will remain "u1". The five RAB units have not changed throughout this process and still have a one to one correspondence with the index. This is not the case with the RMB1ER units. The relationship between these units and the index is no longer one to one. There is no assurance that the original five RMB1ER units remain in the MIN_DIST data set. All that we know is that at least one of the RMB1ER units remains in the data set. So what should the RMB1ER index be? If we use "u2" again it will mean two different things in the three functions. The correct answer may be to introduce a new index "u3". We were unsure so we stayed with the "u2" index.

Step 2: The establishment of the elemental detail tables is easy. It will be exactly the same size as the MIN_DIST table. In this case the third column will contain a flag indicating true, if the RMB1ER unit is in motion, or false if it is not.

Step 3: On the surface the function rule seems simple, and it is if the assumptions we have made are accurate.

Resulting Genus Paragraph:

MOVING_MIN(MIN_DISTu1u2, MOTIONu2)/t/ {MIN_DIST}
; a if(MOTIONu2 = TRUE), true, false) If the RMB1ER unit paired with the RAB unit is moving then MOVING_MIN is true. This calculation is done for each RAB.

Comments: Several assumptions were made in creating this genus. First, we assumed that "u2" was an accurate index for the RMB1ER in the MIN_DIST data set. Second, we brought in the MIN_DIST data set but did not use the value in that data. Instead, all we used was the unit pairing information which appears in the key to the elemental detail table. This pairing information generated a index into the attribute MOTION by taking the index to the RMB1ER and extracting the motion information on that unit. This does not seem like good modeling and we have no idea if it would work.

This question of the index for the RMB1ER units was also complicated by the fact that the value in the MIN_DIST data set was in fact the actual index location of the unit in the elemental detail table. There should have been some way to use this index value to access the motion information. This would have made the model seem more inline with correct modeling, but we did not know how to do this.

Again the question of using the function genus in the index set statement comes up. Here it very nicely defines the elements required for the elemental detail

table. However, if this is illegal, you would have to again revert to the less informative: **Select {LARGE_UNIT} covering u1.**

The fourth task in this program is to examine the 5 X 3 data set generated by MOVING_MIN. This data set consists of a RAB,RMB1ER pair and a flag. If the flag was true then the RMB1ER unit was in motion and the two units were paired. If the flag was false then a new pairing must be sought. Ideally we would like a 5 X 3 data set with the first column being the RAB unit and the second column being the pairing unit, from MOVING_MIN or the new pairing, and the third column being another flag showing if these are good pairings. In a very high level pseudo-code this would look like the following.

```
for u1 = 1 to 5 do
 if u1u2 = false    (u2 is stopped)
 then
    u3 = most advanced Red Man Bat 1st Echelon
    u4 = most advanced Red Man Bat
    if u3 = u4    (unit has not been passed through)
    then
       u2 = u3    (change pairing)
       flag = true
    else    (unit has been passed through)
       if u3 MISSION = attack
       then
          u2 = u3    (change pairing)
          flag = true
       else
          flag = false    (no pairing possible)
       endif
    endif
 endif
enddo
```

At this point in the modeling effort we were stopped. There do not seem to be any tools in the generic rule syntax which would allow the index manipulation shown in the pseudo-code. The ability to conditionally access the rows of the

elemental detail tables and substitute index u3 for u2 does not seem to be provided for. So the ideal 5 X 3 data set with the overall resultant pairs and the boolean flag does not seem achieveable.

There is probably a way around this limitation by using more functions to build more data sets and then having another function review all of these data sets. Some of the logic in the pseudo-code could then be placed in the generic structure. However, at this point we stopped our attempt at using the syntax suggested by Geoffrion. Although we are convinced that the syntax for the generic rule section and the index set statements can be used to construct the required model structure, we were not having much success with it. To continue building this tenuous house of cards with its anemic index set statements, very questionable generic rules and doubts about the correct indices, seemed counterproductive. To our perspective the point had been made. The tools are available but not necessarily appropriate for modeling combat simulation models and not very easy to use.

b. Recommended Alternatives for Logic Representation

There are two possible solutions for the logic programming issue: training or a modification to SM. The training approach might be the simplest course of action but may not be the best. A modification to SM may have considerable impact on SM but the resulting system might be more applicable to simulation modeling.

We have tried to point out that SM has a logic programming capacity of great capability and complexity. We believe that all aspects of the ONEC model could be modeled using SM; even though we could not do so. The obvious answer is training.

Part of the problem, as mentioned before, is the lack of complicated examples to mimic, tutorial texts to review and a workable SM system to experiment with. As SM matures these things will become available. "Programmers" will be able to learn SM and become proficient with the tools.

This answer only addresses part of the problem, programmer training. It does not address the question of how suitable SM tools are for the logic found in simulation systems. The example provided showed some of the problems encountered when trying to use these tools in this domain.

The second solution, one we feel would greatly enhance the applicability of SM to simulation systems, is to modify the syntax for the index set statements and generic rules to incorporate a high order language (HOL) capability. This solution addresses both the training and the suitability problems.

It can be assumed that the simulation modeling will be done by simulation "programmers". These programmers may or may not have a good solid math background; however, they all should have a good solid background in HOL programming. This does not eliminate the training problem; it just reduces its scope. The programmers will still have to learn SM but the hardest part of this, the syntax of the index set statements and the generic rules, will have been greatly simplified.

The syntax for the index set statements could be greatly enhanced by using one of the predicate calculus based programming languages. We understand that this is currently under investigation by Mr. Srikanth Chari, one of Geoffrion's doctoral students. Mr. Chari is investigating the use of Prolog. Another option might be the use of a database query language like SQL. Either of these two options should make the index set statements more readable, easier to program, possibly more descriptive and very conducive to a computer implementation.

The syntax for the generic rules requires a language such as Pascal to handle the problems we've encountered. There is little question that this could provide most capabilities that a modeler might need. It also has the benefit of being something readily understood by the potential modelers. The pseudo-code example shows where a HOL can comfortably handle something which is hard to manage using current SM tools.

This is not a trivial change to SM and may not even be possible. On the surface it seems to avoid some difficult aspects of SM and to provide a capability which more people could understand and use. However, many questions remain to be answered before this could be implemented.

First, is this technically feasible? Can HOLs be integrated into the SM framework without destroying the very solid theoretical foundation which Geoffrion has built? Can the interfaces between the indexing scheme, elemental detail tables, index set statements and the generic rules be worked out and still retain the benefits of SM and the HOLs? In other words, it will not be of any use if the HOLs or SM must be greatly modified.

Second, what is the impact of doing this on the SM products discussed in Chapter IV? Would the greatly increased power in the generic rules tend to detract from the information in the generic graph? Would this cause a migration of the logic currently coded in the generic structure into the function genera? How would this scheme affect the role of the solver? Would a second level of documentation be required for these new powerful logic tools?

We do not have the answers to these questions. They will require extensive study by persons very knowledgeable in SM and computer languages. Our experience in attempting to model ONEC using the current SM tools suggests that it would be a very valuable undertaking. The different presentation of the simulation program data and the manipulations of that data available through SM are definitely worth pursuing.

C. MINOR PROBLEMS

The problems in this section are ones which provided us challenges in our modeling effort but were eventually solved. They are indicative of problems faced by an unsophisticated modeler dealing with complex new tools and limited documentation. Problems of this type are those which we would expect to resolve themselves as SM matures.

1. Problems with Attributes

The rules governing the use of attributes limit the options available to the modeler. They sometimes force the modeler to make decisions which hide information or make unnatural use of the SM elements in order to circumvent these restrictions. They also seem to prevent the logical modeling of attribute inheritance. The following sections will deal with specific examples of problems encountered. Before discussing these specific examples a brief summary of the attribute rules is in order.

1. An attribute cannot call a function or test element [Ref. 1: pg. 2-2].
2. A primitive entity cannot call an attribute [Ref. 1: pg. 2-3].
3. A compound entity cannot call an attribute [Ref. 1: pg. 2-2].
4. An attribute cannot call an attribute [Ref. 1: pg. 3-2].
5. A function cannot call an attribute [Ref. 1: pg. 2-3].
6. An attribute may call a primitive or compound entity [Ref. 1: pg. 2-3].
7. An attribute may call several primitive and or compound entities [Ref. 1: pgs.2-78 and 2-83].

These rules are shown graphically in Figure 5.3.

2. Using Compound Entities in Place of Attributes

A basic theme in this Section concerns the limitations of the attribute element type and ways around these restrictions. A technique which shows up with great regularity is replacing attribute elements with entities. This works because the compound entity elements are not prohibited from calling other entity elements and attributes are allowed to call entities. This circumvents the primary problem of an attribute being unable to call another attribute.

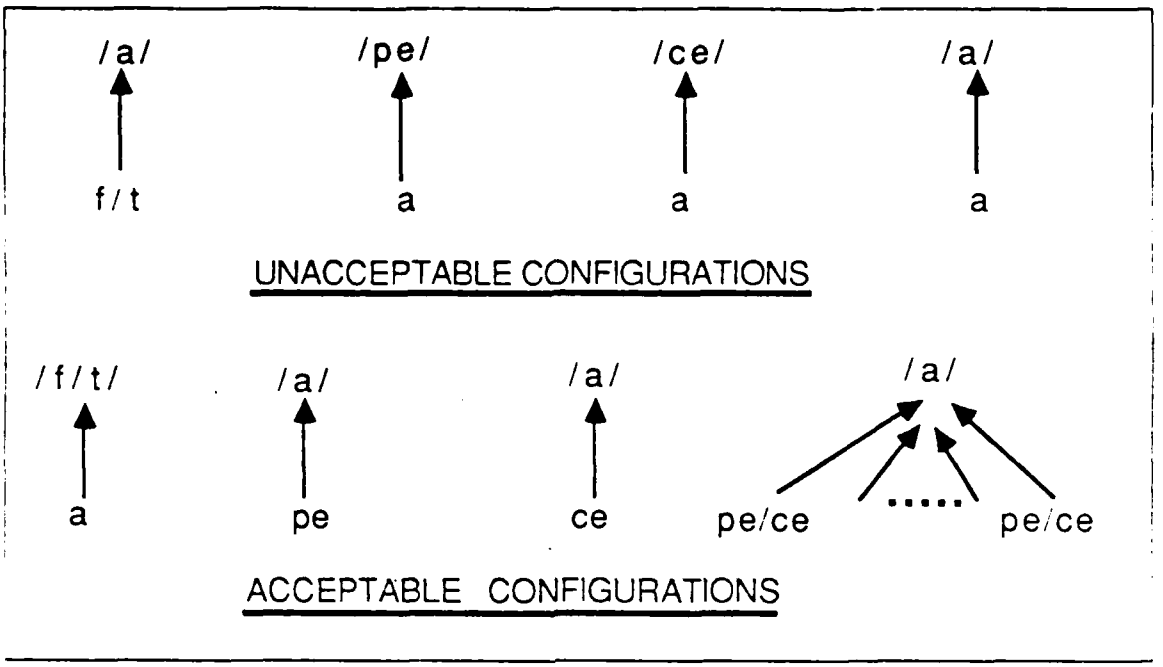


Figure 5.3 Attribute Rules.

This leads to some conceptual problems with SM. Remember that an entity element can be thought of as a "thing" and an attribute is a property of that "thing". This seems straightforward and easy to implement. We can look at something and know it is a "thing" and belongs in an entity element. We can look at something and know it is a property and belongs in an attribute element. But now in the modeling phase we find cases where the model will not work with the simple straightforward allocation of elements to the entity and attribute genera. We are forced to go back into the model and redefine attributes as entities to form a workable structure.

On the surface this seems to be a weakness in SM. In fact this schizophrenic behavior of attributes is discussed by Geoffrion. He states:

> A member attribute for a class can be rendered in SM either as (i) an attribute genus whose elements are 1:1 with the elements of the genus it calls (e.g., HULL_NO), or as (ii) a compound entity genus that links entity elements to elements of some other entity genus that is self-indexed (e.g. TYPE) [Ref. 5: pgs. 2.3].

Ignoring the conceptual problems of an attribute being classified as an entity, something which seems to be endorsed by Geoffrion, the techniques and tools seem to be available to build the required modeling structures. Some examples follow.

The function SPEED_FAC_CELL derives its value from a table search using the current values of RELIEF and VEGETATION. It seems logical that there should be a way to place this table into the model in the form of a genus and access it with the function statement. Several different methods were tried before a workable solution, one which followed the rules of SM, was found. The options considered and a discussion of why they would or would not work follows.

The function is a simple table search using the values of RELIEFg and VEGETATIONg as indices to the table. Given that there are 4 possible relief types and 11 possible vegetation levels this table would contain 44 entries, one for each possible RELIEF/VEGETATION combination. So for a certain GRID_CELLg the values of RELIEFg and VEGETATIONg are used to examine the table and extract the speed factor for that grid cell.

The first method tried was to place the table in an attribute type genus. This is consistent with the recommendations of Geoffrion. He states, "Most of the "coefficients" and "data" of conventional models are represented as attribute elements" [Ref. 1: pg 2-3]. This did not seem to work. The table must be keyed on the relief and vegetation values. This means that the table attribute must have these two attributes in its calling sequence but SM rules prohibit an attribute from calling an attribute.

The option of specifying this table as an attribute of the primitive entity GRID_CELL also does not work. An attribute is used to assign values to elements in a genus. So, for an attribute to define a genus it must have a value for each element in that genus. The GRID_CELL genus has 1610 elements. The table will have 44. There is no way to consider the table as an attribute of the grid cell. It is the function SPEED_FAC_CELL which associates these 44 values with each of the 1610 grid cells.

A workable option is to code the table into the function element. This can be accomplished by using a large case statement with 44 selections. This hard codes the data into the model instead of treating it as data. Although this would work it is extremely awkward and violates good modeling practices.

Another approach is to establish two new primitive entity genera which contain the values for the relief and vegetation attributes. There is a genus attribute called by an attribute genus which combines the two entities using a cartesian product and assigns a value to each resultant pair. This creates the table with the two required key values, all in a manner acceptable to SM. The required genus paragraphs are as shown in Figure 5.4. The graphic representation of this genus structure is shown in Figure 5.5.

GRID_CELLg/pe/

GRID_RELIEF(GRID_CELLg)/a/

GRID_VEG(GRID_CELLg)/a/

RELIEFr/pe/
There is a list of all relief values.

VEGv/pe/
There is a list of all vegetation values.

SPEED_FAC_TABLE(RELIEFr, VEGv)/a/ {RELIEF} X {VEG}
There is a speed factor for every combination of relief and vegetation.

SPEED_FAC_CELL(GRID_RELIEFg, GRID_VEGg, SPEED_FAC_TABLEr,v)
/f/ ; Select {SPEED_FAC_TABLE}
Where VEGg = GRID_VEGg and RELIEFr = GRID_RELIEFg

Figure 5.4 Genus Paragraphs for Table Model

This approach is a good one because it allows the data to be treated as data instead of being inserted into the code of a function. It also adheres to the rules of SM. This may not be an immediately obvious approach nor particularly elegant or convenient use of the SM element types, but it works where other approaches failed.

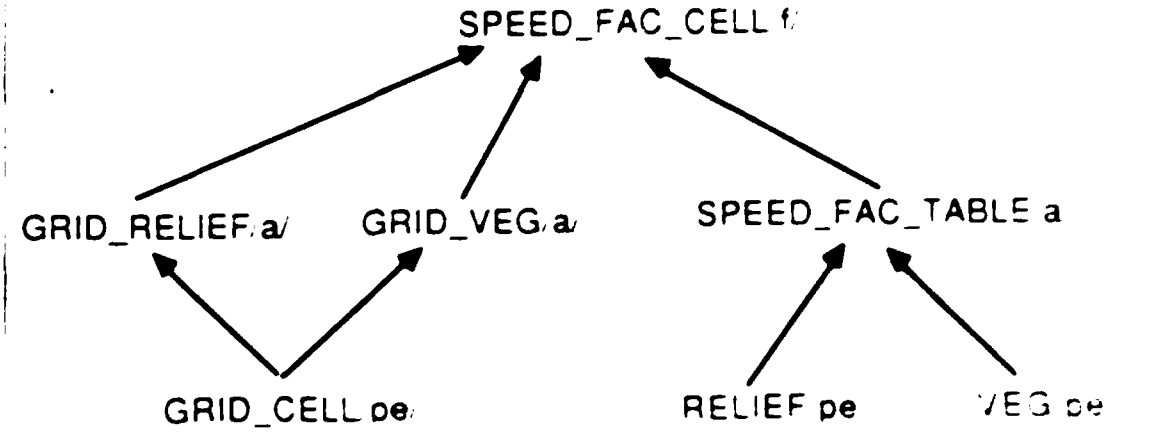


Figure 5.5 Table Model

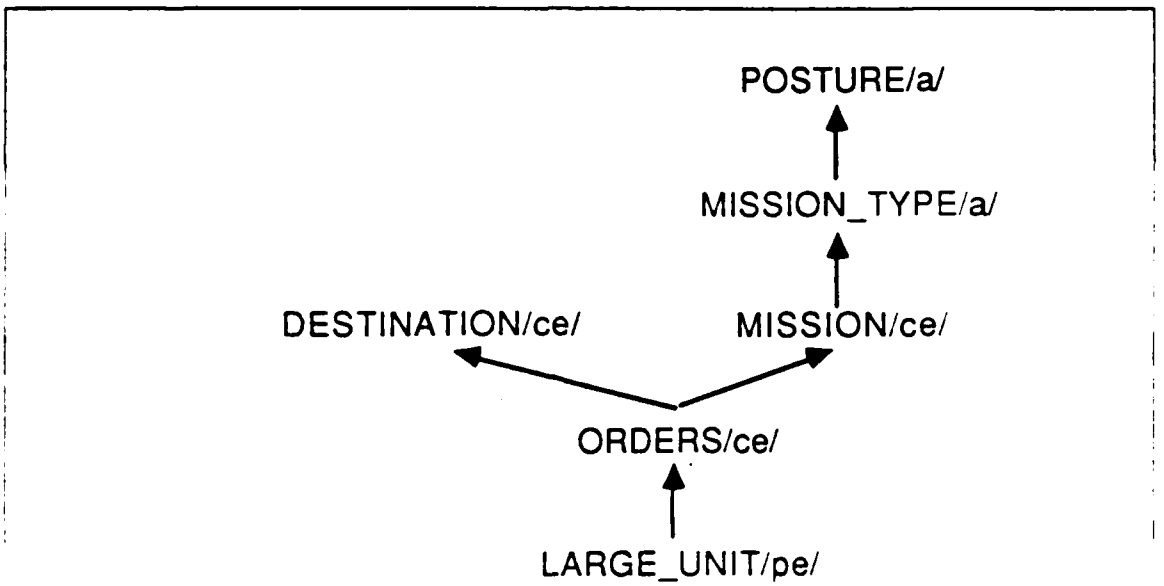
Another example can be found in Geoffrion's work on Hammer and McLeod's Tanker Modeling Database. In this work Geoffrion models the attribute ship type as a primitive entity TYPE_OF_SHIP and a compound entity TYPE, which calls the primitive entities SHIP and TYPE_OF_SHIP. This seems to have been done to mimic the organization of the Semantic Data Model, rather than to model around the restrictions posed by the use of attributes. This example is provided just to show that it is acceptable to model an attribute as an entity if required. [Ref. 5: pgs. 8-9]

A final example of this problem stems from a situation where an attribute defines another attribute. This comes about in the section of the model dealing with the orders. Each set of orders has a mission and a destination, each mission has a mission type and one mission type has a set of three postures. The obvious, but incorrect approach, is to model the orders and missions as compound entities and the mission type and postures as attributes (Figure 5.6). This again is not allowed because an attribute may not call an attribute.

One way around this is to build a primitive entity MISSION_TYPES and a compound entity MISSION_TYPE which would call both MISSION and MISSION_TYPES. POSTURE could then remain as an attribute to MISSION_TYPE as shown in Figure 5.7. This may work. It is somewhat awkward but it does retain all of the information and shows the relationships between the mission type and the posture. However, it does require the introduction of a seemingly unnecessary primitive entity. The primary objection to this method is that the compound entity MISSION_TYPE is not variable and this model requires that a unit be able to change missions as the simulation progresses. So it seems that the method of modeling an attribute with a primitive and compound entity combination will not work when trying to replace a variable attribute.

Our final choice was to define the mission entity as a variable attribute with a range which included every possible mission type and posture. This approach does not show graphically the relationship between the mission type and the postures but it does provide the information in the genus text. It also solves the problem of the changing mission and reduces the required number of generators by three. This approach is shown in Figure 5.8.

This example was chosen to demonstrate the difficulties a modeler may face in [illegible] structured model. Here we have gone full circle from an attribute to a [illegible] and back to an attribute.



LARGE_UNITu /pe/

ORDERS(UNITu) /ce/ {LARGE_UNIT}

DESTINATION(ORDERSu) /ce/ {LARGE_UNIT}

MISSION(ORDERSu) /ce/ {LARGE_UNIT}

MISSION_TYPE(MISSIONu) /va/ {LARGE_UNIT} (RED MISSIONS attack, holding attack, be prepared to attack; BLUE MISSIONS delay, withdraw, reserve, move to reinforce, defend)

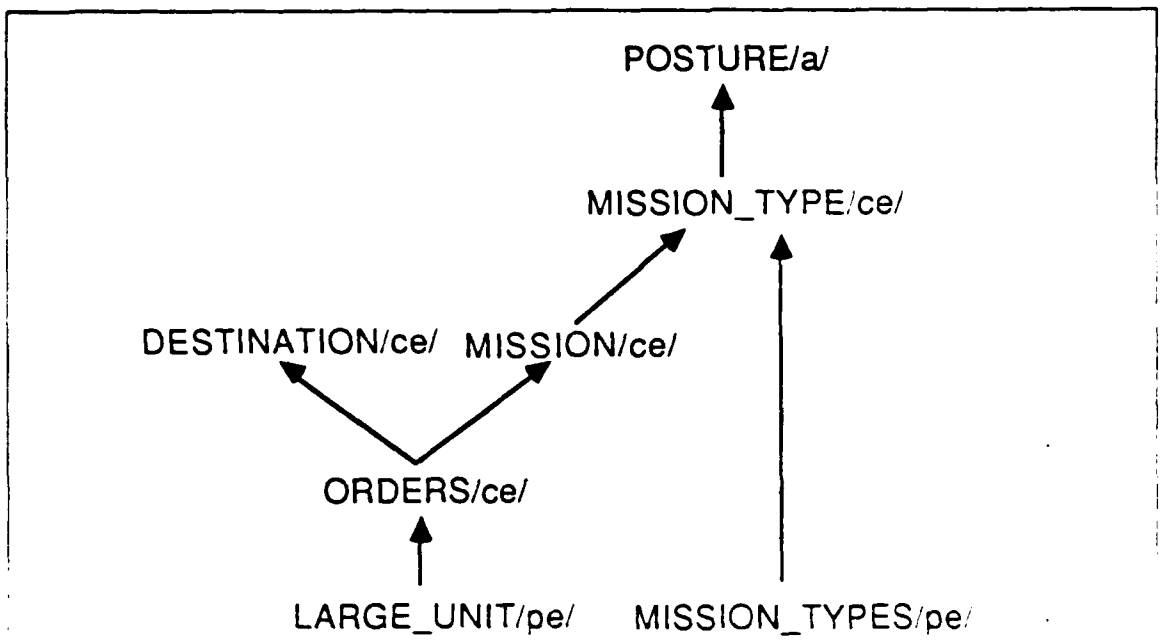
POSTURE(MISSION_TYPEu)/va/ Select{LARGE_UNIT} Where MISSION_TYPE = "DEFEND (fortified position, hasty defense, prepared position) These are the postures for the mission type defend

Figure 5.6 Improper use of Attributes.

3. **Abstract Data Types**

There does not seem to be a capability , in SM, to build a data type which can be applied to more than one genus while still addressing a single genus. This capability would have been very useful when dealing with aspect of location and the table issue. This is a minor inconvenience which can be avoided with resourceful modeling. The table example was covered in sufficient detail in the last Section. Location is addressed below.

Three of the primitive entities in the model, GRID_CELL, LARGE_UNIT and SMALL_UNIT, require information about their location. In all three cases this information can be modeled as 2 sets of (X,Y) coordinate pairs with identical range requirements. This suggests that a single data type could serve for all three entities.



LARGE_UNITu /pe/

ORDERS(UNITu) /ce/ {LARGE_UNIT}

DESTINATION(ORDERSu) /ce/ {LARGE_UNIT}

MISSION (ORDERSu) /ce/ {LARGE_UNIT}

MISSION_TYPESm/pe/ There is a list of all mission types.

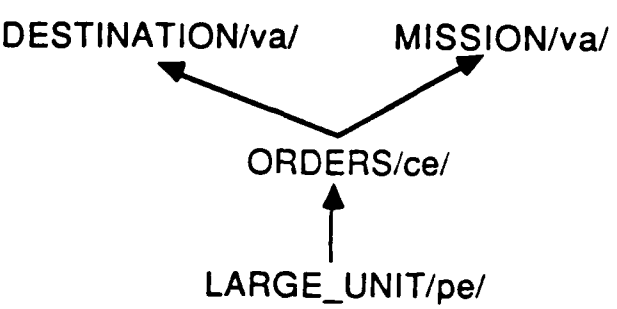
MISSION_TYPE (MISSIONu, MISSION_TYPESm) /ce/ {LARGE_UNIT}

POSTURE(MISSION_TYPEu)/va/ Select{LARGE_UNIT} (fortified position, hasty defense, prepared position) These are the postures stated for the mission of defend.

Figure 5.7 Modeling Attributes as Compound Entities.

The first option considered was to have a single attribute called LOCATION which could be used whenever location information was required. At first this was rejected because the excess baggage in generic calling sequence and index set statement seems to defeat the intent. The resulting statement looked like:

LOCATION (GRID_CELLg, LARGE_UNITu, SMALL_UNITs) /va/Select(GRID_CELL, LARGE_UNIT, SMALL_UNIT): 0 < = X < = 135, 0 < = Y < = 135



LARGE_UNITu /pe/

ORDERS(UNITu) /ce/ {LARGE_UNIT}

DESTINATION(ORDERSu) /va/ {LARGE_UNIT}

MISSION (ORDERSu) /va/ {LARGE_UNIT} (attack, holding attack, be prepared to attack, delay, withdraw, reserve, move to reinforce, defend fortified position, defend hasty defense, defend prepared position)

Figure 5.8 Current Approach to Modeling the Mission Attribute

After further thought it is not clear that this approach would have worked even if we had been willing to accept the impact of the excess baggage. Although attributes can call more than one entity element this approach would not have had the desired effect. When attributes call more than one entity they are providing a value to the combination of those entities. This is exactly the approach used in the solution to the table issue with the attribute SPEED_FAC_TABLE. The desired goal was to have the attribute apply to the entities one at a time, but this does not seem possible.

To work around this problem the current approach is to use a different attribute for the LOCATION of each item. So the model now has attributes for LOC_LARGE_UNIT, LOC_GRID_CELL and LOC_SMALL_UNIT. This tends to run contrary to the concept of aggregation, however it works.

4. **Inheritance**

Geoffrion does not explicitly state how inheritance issues are resolved in SM. We examined several possibilities and reached a conclusion on what we thought was the best solution for modeling inheritance within the SM framework. The options considered and a discussion of their merits and weaknesses follows.

The first alternative was to show inheritance explicitly through the generic calling sequence. The underlying intent was to have the model show exactly which attributes were inherited and which were not. This we felt would go a long way in helping a user to understand the underlying element relationships in the program. Three model structures were considered.

In the following examples a simple scenario is used. There is a primitive entity called PE1. It has two attributes: A1 and A2. CE1 is a compound entity which is a subset of PE1. CE1 will inherit attribute A2 but not attribute A1. In the final example CE1 has an attribute A3 and a compound entity CE2, which is a subset of CE1, and PE1 has an additional compound entity CE3.

The first model structure considered is shown in Figure 5.9. Graphically this looks very nice. It is easy to see the exact relationship which exists between CE1 and the two attributes A1 and A2. But this approach does have a fatal flaw; it is an illegal structure in SM. A compound entity may not call an attribute. So this option was rejected.

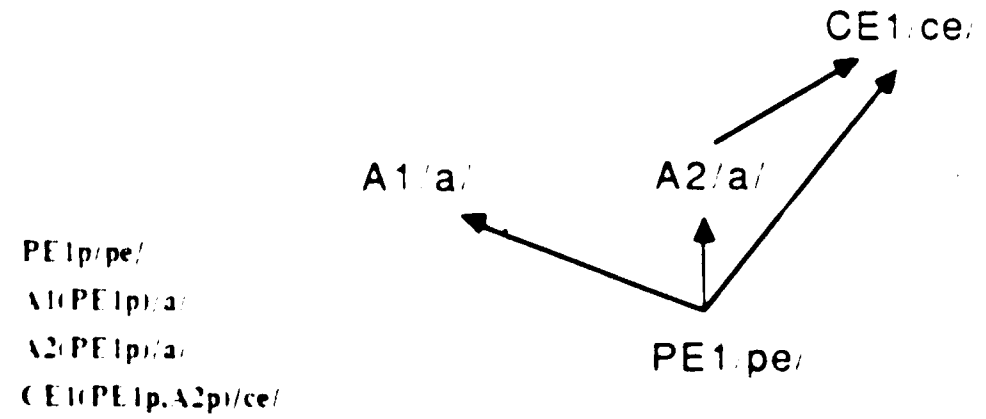


Figure 5.9 Inheritance Approach 1

The second model structure considered is shown in Figure 5.10. Again the graphical structure shows the essence of the relationship between CE1 and the two attributes. However, just as this is an illegal structure in SM, it is not even [illegible] PE1 [illegible]

For A2 to be used in this manner PE1 and CE1 would have to have a 1 to 1 correspondence because A2 would have both PE1 and CE1 in its calling sequence. Obviously this is not going to help; so this option was also rejected.

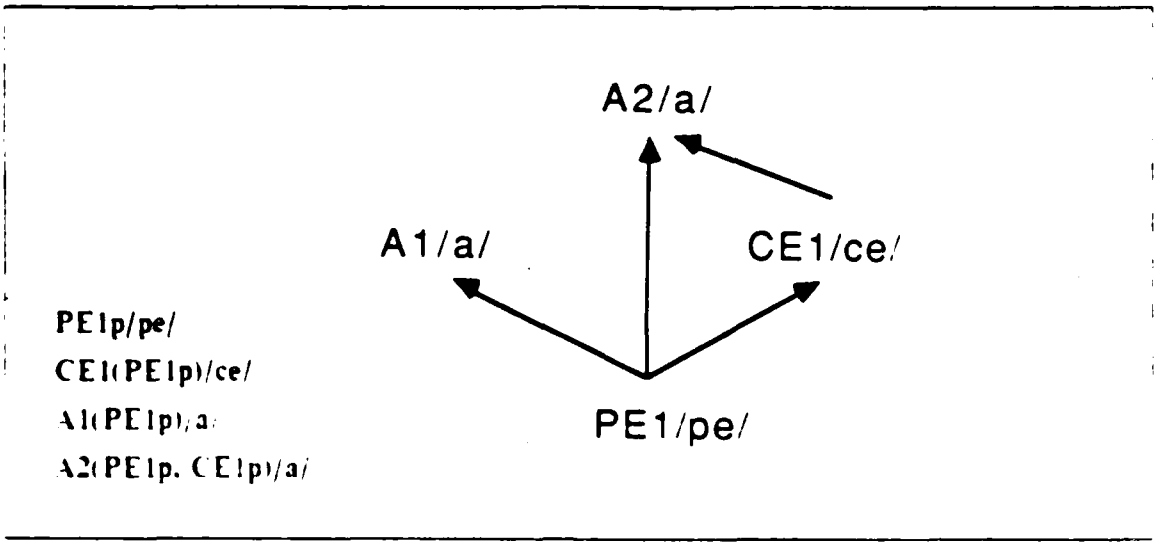


Figure 5.10 Inheritance Approach 2

The third and last option considered in the search for explicit inheritance is shown in Figure 5.11. This approach also shows the relationship between CE1 and A2; however, it makes no sense to have the same attribute in two different locations and it is not legal in SM. It is illegal to have the same attribute in two different locations with two different generic calling sequences and two different index set statements. So, it seems there may not be a way to model inheritance explicitly in SM. How then, is it done?

Since explicit inheritance seems impossible to model in SM then it must be assumed that some sort of default inheritance is in existence. We assume this means that every compound entity assumes all of the attributes of every related entity below it in the generic graph. A related entity is one which appears either directly or indirectly in the compound entity's calling sequence.

It is easy to see how such an approach would work. Remember how the elemental detail tables were constructed using the indices in the generic calling sequences as the keys to the tables. If a certain compound entity has a certain entity's index in its calling sequence then it would have a logical path to the elemental detail table that entity was stored to access all of the attributes of that entity.

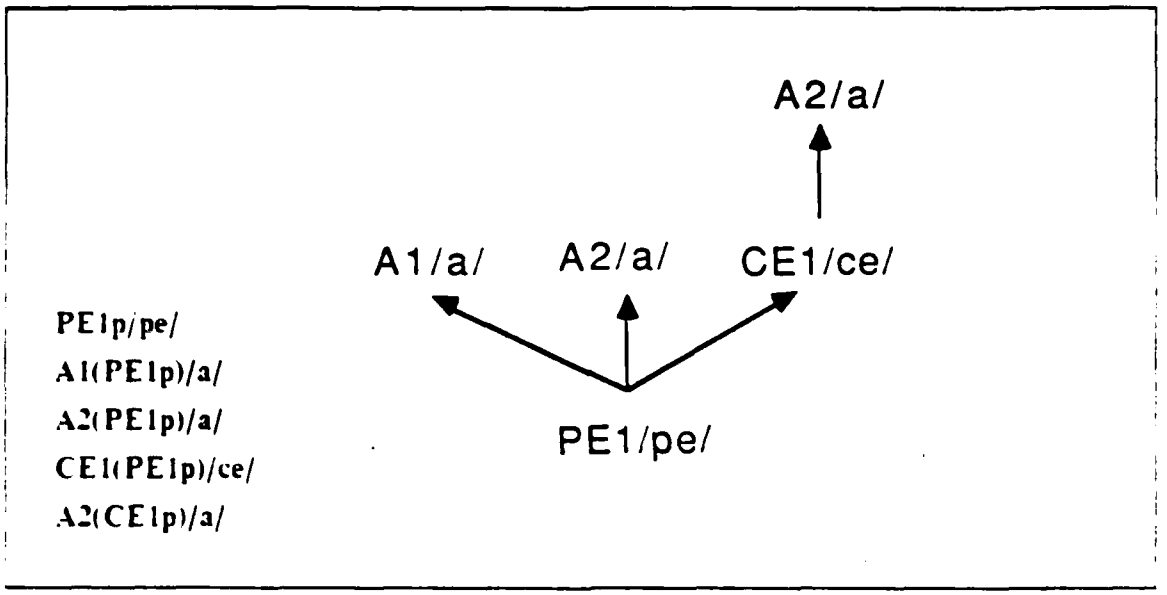


Figure 5.11 Inheritance Approach 3.

In the absence of specific guidance on this issue we assume that the default inheritance procedures defined above are acceptable SM modeling practice. Figure 5.12 shows this concept.

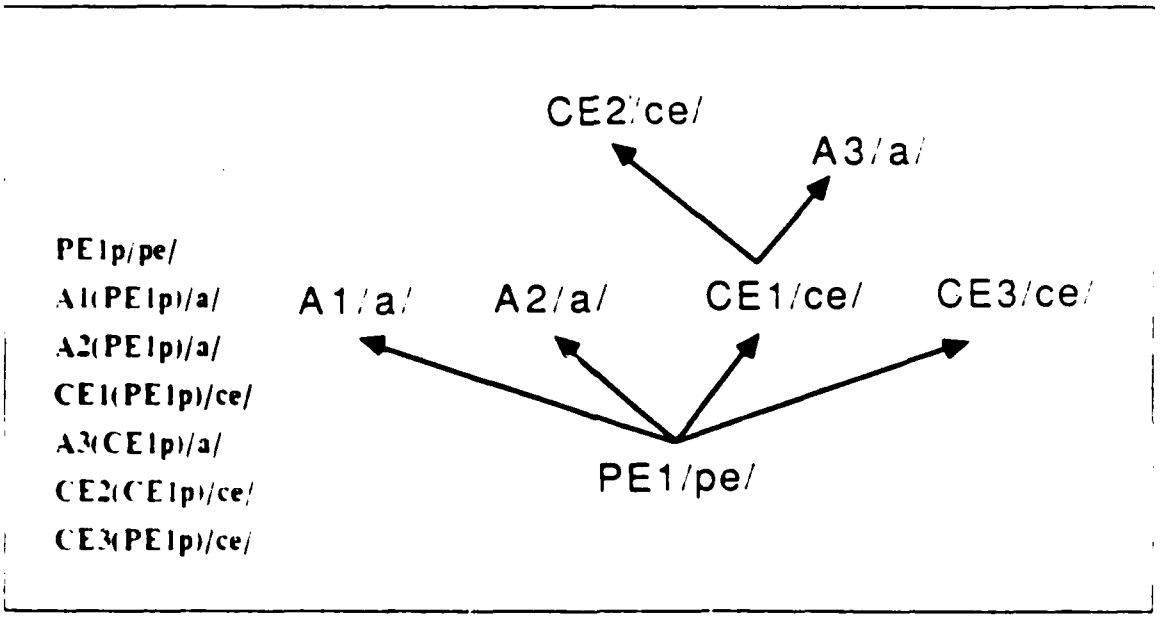


Figure 5.12 Inheritance Chosen Solution.

From Figure 5.12 we assume that CE1 would inherit both A1 and A2 from PE1. CE2 would inherit A1 and A2 from PE1 and A3 from CE1. CE3 would inherit A1 and A2 from PE1 but would not inherit A3 from CE1.

This concept of inheritance will play a major role in the discussion of modeling hierarchies which is the topic of the next Section.

5. **Hierarchy of Units**

The LARGE_UNIT elements in the ONEC model exist within a hierarchal structure. To define a LARGE_UNIT's position in this structure you must know its LEVEL, (division,regiment, or battalion), its ECHELON, (1st, 2nd or reserve), and its TYPE, (maneuver or artillery). An example of the general structure is shown in Figure 5.13.

```
1st Echelon Division
    1st Echelon Regiment
        1st Echelon Maneuver/Artillery Battalion
        2nd Echelon Maneuver/Artillery Battalion
    2nd Echelon Regiment
        1st Echelon Maneuver/Artillery Battalion
        2nd Echelon Maneuver/Artillery Battalion
    Reserve Regiment
        Reserve Maneuver/Artillery Battalion
2nd Echelon Division
    1st Echelon Regiment
        1st Echelon Maneuver/Artillery Battalion
        2nd Echelon Maneuver/Artillery Battalion
    2nd Echelon Regiment
        1st Echelon Maneuver/Artillery Battalion
        2nd Echelon Maneuver/Artillery Battalion
    Reserve Regiment
        Reserve Maneuver/Artillery Battalion
```

Figure 5.13 Hierarchy in ONEC.

Several different options were considered for modeling the hierarchy, yet none of them seemed exactly right. In the end it was decided that because ONEC did not use the hierarchy information, it was not essential to model it. In the current approach the hierarchy information is placed in the LARGE_UNIT_TYPE genus paragraph as shown below.

LARGE_UNIT_TYPE(LARGE_UNITu) /a/ {LARGE_UNIT}: (List all unit types)
Every UNIT has a description which fully defines that UNIT in the Army hierarchy. This will include the LEVEL of the UNIT (Division, Regiment, Battalion) the ECHELON of the UNIT (First, Second, Reserve) and the TYPE of the UNIT (Artillery or Maneuver).

This approach provides the necessary hierarchy information while avoiding the issues of modeling the hierarchy and the related issue of attribute inheritance. Notice how the information is hidden in the text and unavailable in the graphical presentation. Also note that every unit must share all of the same attributes. This avoids the problem without providing an answer.

It will be necessary to find an acceptable SM representation for this hierarchy if SM were to be applied to a more complicated model, such as FOURCE, where the hierarchy information plays an important role. We were unable to develop an acceptable model on our own. However, Geoffrion has recently released an informal note "Modeling Categorization Hierarchies" [Ref. 13]. In this work Geoffrion describes and comments on five different approaches to this issue. In the following sections each of these five suggested approaches will be applied to the ONEC hierarchy and comments provided on the merits of each. To enhance understanding of these five approaches each section will start with a quote from Geoffrion's work describing the approach.

a. Approach 1

> One rather obvious idea is to design the schema so that the modular structure (which, of course, is always a tree) mimics exactly the categorization hierarchy. That is, we want the modular tree to be isomorphic to the categorization hierarchy tree. In order for this to be so, modules should be 1:1 with categories and genera 1:1 with items. [Ref. 13: Pg. 3].

This is very easy to implement. The resulting schema is shown in Figure 5.14. Notice in the notation of Figure 5.14 that the primitive entities would have to be numbered to reflect the individual units. This is shown with an 'N' where the actual number would go. This Figure has been simplified by removing the 2nd Echelon Division information. This information is essentially a duplicate of the 1st Echelon Division information with 2nd in place of 1st.

This approach does not show any information in the generic graph. It would just look like isolated nodes; one for each unit in the model. All of the information would show up in the modular structures and module graphs. Geoffrion also points out that this approach would generate a large schema for hierarchies with a large number of items [Ref. 13: Pg. 5].

In this case this limitation seems to be fatal. It would be impossible to treat each unit in the simulation as an individual genus. This approach would also

```
&1ED  1st Echelon Division
   &1ED1ER 1st Echelon Regiment of 1ED
      &1ED1ER1EB 1st Echelon Battalion of 1ER of 1ED
         1ED1ER1EB_MANEUVER_1  pe
         1ED1ER1EB_MANEUVER_2  pe
               :       :
         1ED1ER1EB_MANEUVER_N  pe
         1ED1ER1EB_ARTILLERY_N  pe
      &1ED1ER2EB 2nd Echelon Battalion of 1ER of 1ED
         1ED1ER2EB_MANEUVER_N  pe
         1ED1ER2EB_ARTILLERY_N  pe
   &1ED2ER 2nd Echelon Regiment of 1ED
      &1ED2ER1EB 1st Echelon Battalion of 2ER of 1ED
         1ED2ER1EB_MANEUVER_N  pe
         1ED2ER1EB_ARTILLERY_N  pe
      &1ED2ER2EB 2nd Echelon Battalion of 2ER of 1ED
         1ED2ER2EB_MANEUVER_N  pe
         1ED2ER2EB_ARTILLERY_N  pe
   &1EDRR Reserve Regiment of 1ED
      &1EDRRRB Reserve Battalion of RR of 1ED
         1EDRRRB_MANEUVER_N  pe
         1EDRRRB_ARTILLERY_N  pe
```

Figure 5.14 Hierarchy Approach 1.

raise problems with attributes. There is no way to have a single attribute for all of the units. This would have to be placed in the module paragraph description, which means that the information would not show up in the elemental detail tables, or in individual attribute would have to be created for each genus.

b. Approach 2

An alternative design objective is to craft the schema so that the module structure mimics only the category tree rather than the full category/genus hierarchy. That is, we want the modular tree to be essentially a simplified [illegible]

category tree. Items and their associations with categories are to be reflected in generic or elemental structure. In order to accomplish this, each category that is not a leaf of the category tree should correspond to a module, and each category that is a leaf of the category tree should correspond to a genus. [Ref. 13: Pg. 5]

This is also fairly straightforward and is shown in Figure 5.15

LARGE_UNITu pe

&1ED 1st Echelon Division

&1EDIER 1st Echelon Regiment of 1ED

&1EDIER1EB 1st Echelon Battalion of 1ER of 1ED

1EDIER1EB_MANEUVER LARGE_UNITa ce

1EDIER1EB_ARTILLERY LARGE_UNITa ce

&1EDIER2EB 2nd Echelon Battalion of 1ER of 1ED

1EDIER2EB_MANEUVER LARGE_UNITa ce

1EDIER2EB_ARTILLERY LARGE_UNITa ce

&1ED2ER 2nd Echelon Regiment of 1ED

&1ED2ER1EB 1st Echelon Battalion of 2ER of 1ED

1ED2ER1EB_MANEUVER LARGE_UNITa ce

1ED2ER1EB_ARTILLERY LARGE_UNITa ce

&1ED2ER2EB 2nd Echelon Battalion of 2ER of 1ED

1ED2ER2EB_MANEUVER LARGE_UNITa ce

1ED2ER2EB_ARTILLERY LARGE_UNITa ce

&1EDRR Reserve Regiment of 1ED

&1EDRRRB Reserve Battalion of RR of 1ED

1EDRRRB_MANEUVER LARGE_UNITa ce

1EDRRRB_ARTILLERY LARGE_UNITa ce

Attributes which apply to all units can be handled easily by developing an attribute which calls the primitive entity LARGE_UNITS. Then all of the compound entities will inherit these attributes as discussed in the last section. Attributes which apply to an entire division, regiment, or battalion cannot be handled formally. These categories of the hierarchy are modeled using the modular structure and have the same problems with attributes as the first approach. Attributes which apply to units at the lowest level of the hierarchy, i.e. Maneuver Units of the 1st Echelon Division 1st Echelon Regiment 1st Echelon Battalion, can be handled formally. This is easy to do because the bottom of the hierarchy is modeled using compound entities and attributes can call compound entities.

c. Hierarchy Approach 3

> A third approach is like the first, except that the generic structure rather than the modular structure is used to mimic the categorization hierarchy. We desire the genus graph, rather than the modular tree, to be isomorphic to the categorization hierarchy tree. [Ref. 13: Pg. 7]

This approach, shown in Figure 5.16 , is a simple translation of the first approach shown in Figure 5.14. The 1st and 2nd Echelon Divisions are represented by primitive entities and everything else uses compound entities to form the different echelon levels. Again, there are problems with the size of the resulting generic structure and with attributes.

This approach requires a separate compound entity for each unit in the model. This was an unacceptable requirement in the first approach and remains unacceptable here. The handling of attributes is better with this approach but still has some problems. For example, it is still impossible to have a single attribute which applies to all units. However, it is possible to have attributes which apply to all units under a certain level of the hierarchy, i.e. all 1st Echelon Division units.

d. Hierarchy Approach 4

> Our design objective now is to devise an approach that is to the second approach as the third is to the first. That is, an approach wherein generic structure rather than modular structure is used to mimic the category tree. Items and their associations with categories are to be represented in elemental structure. Thus we want the genus graph, rather than the modular tree, to be isomorphic to the category tree. [Ref. 13: Pg. 9]

1ED/pe/ 1st Echelon Division
1ED1ER(1ED)/ce/ 1st Echelon Regiment of 1ED
1ED1ER1EB(1ED1ER) ce 1st Echelon Battalion of 1ER of 1ED
1ED1ER1EB_MANEUVER_1(1ED1ER1EB) ce
1ED1ER1EB_MANEUVER_2(1ED1ER1EB) ce
⋮ ⋮ ⋮
1ED1ER1EB_MANEUVER_N(1ED1ER1EB) ce
1ED1ER1EB_ARTILLERY_N(1ED1ER1EB) ce
1ED1ER2EB(1ED1ER) ce 2nd Echelon Battalion of 1ER of 1ED
1ED1ER2EB_MANEUVER_N(1ED1ER2EB) ce
1ED1ER2EB_ARTILLERY_N(1ED1ER2EB) ce
1ED2ER(1ED) ce 2nd Echelon Regiment of 1ED
1ED2ER1EB(1ED2ER) ce 1st Echelon Battalion of 2ER of 1ED
1ED2ER1EB_MANEUVER_N(1ED2ER1EB) ce
1ED2ER1EB_ARTILLERY_N(1ED2ER1EB) ce
1ED2ER2EB(1ED2ER) ce 2nd Echelon Battalion of 2ER of 1ED
1ED2ER2EB_MANEUVER_N(1ED2ER2EB) ce
1ED2ER2EB_ARTILLERY_N(1ED2ER2EB) ce
1EDRR(1ED) ce Reserve Regiment of 1ED
1EDRRRB(1EDRR) ce Reserve Battalion of RR of 1ED
1EDRRRB_MANEUVER_N(1EDRRRB) ce
1EDRRRB_ARTILLERY_N(1EDRRRB) ce

Figure 5.16 Hierarchy Approach 3.

Figure 5.17 shows the schema which supports this approach. This is a slight deviation from Geoffrion's approach in that there is a primitive entity for all [illegible] and the hierarchy is constructed using compound entities. This is almost identical to Approach 2 shown in Figure 5.15. This differs from Geoffrion's suggestion which [illegible] introduced primitive entities at the division level of the hierarchy [illegible]

This looks like a good sound approach. The hierarchy information [illegible] in the generic graphs. The number of generics is large but manageable and attributes can be applied at any level of the hierarchy [illegible] implementation [illegible] provided [illegible] attributes [illegible]

[illegible]

[illegible] the Red and Blue units. The hierarchy [illegible]

e. Hierarchy Approach [5]

> [illegible]
>
> [illegible] detail [illegible] the schema.
>
> [illegible] elements [illegible] the schema [illegible] hierarchy more abstractly. (Ref 13: Pgs [illegible])

The implementation of this approach is a little more complicated than the last four. Figure 5.18 shows the genus paragraphs and the corresponding generic graph for this approach.

Graphically this shows the general hierarchical structure but it does not show the same level of detail that was available in the other four approaches. Specifically, you cannot examine the generic graph and tell that each division is broken down into three echelons of regiments, and so on. All four of the other approaches provided this information in the modular or generic structure. In this approach this level of detail is available only in the elemental detail tables, but it is available.

The attribute issue is handled very well. Attributes can be assigned to all units in general, or to any level of the hierarchy, a very powerful and flexible capability.

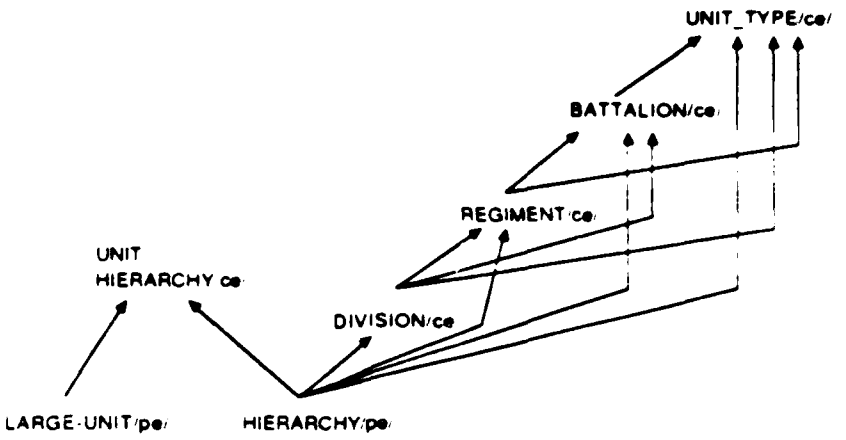
```
LARGE_UNITtype
1ED(LARGE_UNITSource 1st Echelon Division
1EDIER(1EDSource 1st Echelon Regiment of 1ED
1EDIER1EB(1EDIER1Source 1st Echelon Battalion of 1ER of 1ED
1EDIER1EB_MANEUVER(1EDIER1ESource
1EDIER1EB_ARTILLERY(1EDIER1ESource
1EDIER2EB(1EDIERSource 2nd Echelon Battalion of 1ER of 1ED
1EDIER2EB_MANEUVER(1EDIER2ESource
1EDIER2EB_ARTILLERY(1EDIER2ESource
1ED2ER(1EDSource 2nd Echelon Regiment of 1ED
1ED2ER1EB(1ED2ERSource 1st Echelon Battalion of 2ER of 1ED
1ED2ER1EB_MANEUVER(1ED2ER1ESource
1ED2ER1EB_ARTILLERY_N(1ED2ER1ESource
1ED2ER2EB(1ED2ERSource 2nd Echelon Battalion of 2ER of 1ED
1ED2ER2EB_MANEUVER(1ED2ER2ESource
1ED2ER2EB_ARTILLERY(1ED2ER2ESource
1EDRR(1EDSource Reserve Regiment of 1ED
1EDRRRB(1EDRRSource Reserve Battalion of RR of 1ED
1EDRRRB_MANEUVER(1EDRRRBSource
1EDRRRB_ARTILLERY(1EDRRRBSource
```

Figure 5.17 Hierarchy Approach 4.

Geoffrion also points out that this is the most flexible approach of the five when considering possible changes to the hierarchy [Ref. 13: Pg 10]. Certainly this approach isolates the hierarchy model from the rest of the model which should simplify any required changes.

f. Summary of Hierarchy Approaches

Geoffrion's paper proposes five different alternatives for modeling categorization hierarchies. This is by no means an exhaustive list but it shows the complexities facing the modeler when attempting to model a simple structure. The decision on which modeling approach to use is not clear cut and must be evaluated on



LARGE_UNITu/pe/ There is a list of all units.

HIERARCHYh/pe/ There is a list of all possible levels in the hierarchy.

DIVISION(HIERARCHYh)/ce/ Select {HIERARCHY} There are two division echelons: 1st and 2nd.

REGIMENT(HIERARCHYh, DIVISIONh1(h))/ce/ Select ({HIERARCHY} - {DIVISION}) There are three regiment echelons: 1st, 2nd, and reserve.

BATTALION(HIERARCHYh, DIVISIONh1(h), REGIMENTh2(h)) /ce/ Select ({HIERARCHY} - ({DIVISION} - {REGIMENT})) There are three battalion echelons: 1st, 2nd, and reserve.

UNIT_TYPE(HIERARCHYh, DIVISIONh1(h), REGIMENTh2(h), BATTALIONh3(h)) /ce/ Select ({HIERARCHY} - ({DIVISION} - {REGIMENT} - {BATTALION})) There are two unit types: maneuver and artillery.

UNIT_HIERARCHY(LARGE_UNITu, HIERARCHYh4(u)) /ce/ {LARGE_UNIT} Every large unit can be associated with a specific position in the hierarchy.

Figure 5.18 Hierarchy Approach 5.

a case by case basis. For the ONEC hierarchy it seems Approaches 4 and 5 are the best.

Approaches 1 and 3 were designed for systems with very few units in the hierarchy. This is obviously not the case in ONEC, so these two options can be dropped from consideration.

Approach 2 would work with ONEC. The number of genera required is manageable. However, this approach relies on the modular structure to represent the

hierarchy which can cause problems with the use of attributes. Although this would work, there are better options.

Approach 4 must be given strong consideration. It graphically shows the hierarchy in fine detail and provides a very versatile means for applying the attributes. However, it does generate a large generic structure.

Approach 5 does not show the hierarchical structure graphically as well as the other four options. However, this is the result of an attempt to make the hierarchical structure easier to modify by placing the hierarchical structure information in the elemental detail tables [Ref. 13: Pg. 10]. Approach 5 handles attributes just as well as Approach 4 and has a much smaller schema, 7 genera as compared to 39. This approach also seems to be the easiest to integrate into the existing ONEC model.

6. **Indexing**

Structured modeling is based on a generic graph structure. The general relationships which exist between the genera in this graph structure are coded into the generic calling sequence section of each genus. At a finer level of detail it is not just the genus relationships that are shown but actually the element to element relationships which exist between genera. This very fine resolution is made available through a complex indexing scheme; which is a very powerful tool and can be difficult to use. An example which has caused problems in the ONEC model deals with how to index the generic calling sequence of the function genus ROAD_SPEED_FAC.

The function ROAD_SPEED_FAC is responsible for calculating a speed factor for each unit based on the units direction and the availability of roads in the grid cell which the unit occupies. It is easy to identify a single unit, index 'u', or a single grid cell, index 'g', but it is much more difficult to identify a unit and the specific subset of grid cells involved. Our first attempt at the ROAD_SPEED_FAC index calling sequence was as follows:

ROAD_SPEED_FAC(SPEED_FAC_AXIALg, SPEED_FAC_LATERALg, DIRECTIONu)/f/

This would not work because there is no link between the unit and the grid cells. As written every unit and every grid cell would have to be considered. Our second attempt was closer.

ROAD_SPEED_FAC(SPEED_FAC_AXIALg1(u), SPEED_FAC_LATERALg1(u), DIRECTIONu) /f/

This is more along the correct lines. The specific grid cell for a specific unit has now been identified by the index 'g1(u)'. However, is this enough? Where did the pairing 'g1(u)' take place? There is certainly not enough information or logic in this function statement to establish the link. So an additional step must be required to establish the index 'g1(u)'.

We did not attempt to make this additional step. But it would appear that a new compound entity is required to show the pairing of units and grid cells based on their locations:

UNIT_GRID_CELL(GRID_CELLg, LARGE_UNITu) /ce/

This would then lead to a ROAD_SPEED_FAC function statement of:

ROAD_SPEED_FAC(UNIT_GRID_CELLg1(u), SPEED_FAC_AXIALg1(u), SPEED_FAC_LATERALg1(u), DIRECTIONu)/f/

The point to be made is that the modeler must pay strict attention to the indices. He must define the relationships between the elements within the genera and then build the model structure required to develop this relationship. It is not enough to just provide an index. The modeler must provide the logic and structure to support the index.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This was a preliminary attempt at exploring the applicability of structured modeling to the domain of discrete event simulation. We were aware at the onset that SM was not designed for discrete event simulation models and therefore were not surprised that the two domains do not always mesh well. We were unable to consider the important aspect of time in the model because of the complexities of learning SM and ONEC, however we feel that we have learned some important things.

It is our opinion that in its current form SM is adequate to represent simulation models. However, we do not feel that in its current state this would be a productive course of action. There are certain areas which we feel must be addressed before SM can become a productive tool for facilitating discrete event simulation models. We also feel that the benefits provided by using SM provide a powerful incentive to continue the investigation of SM in this arena.

Structured modeling provides a wide variety of very desirable features for a model management environment. These features affect the entire model life cycle including development, use and maintenance. The most significant feature is that SM deals with the entire model and does so in a format which is accessible to both humans and computers.

SM plays a key role in the development phase of a model by encouraging, or at least providing for, good modeling habits. Program development by top down modular design using stepwise refinement is a natural process with SM. In addition, strong data definition and typing is built into the genus paragraphs.

Another aspect which spans the development, use and maintenance phases, is the ability to communicate information about a program at any level of abstraction required. Most software documentation tools deal only with specific aspects of a system and as a rule they do not provide any capability to tailor the presentation of information in a dynamic fashion. SM is much more than just a documentation system. It deals with all aspects of a model, provides numerous different ways to view this information and provides a structure which will allow the user to dynamically alter the presentation's level of abstraction. The result is a powerful tool for model information exchange among the clients, management and programmers.

Two key tasks in software maintenance deal with understanding a program and being able to track the implications of a change to a part on the whole. The SM presentations of the generic structure aid these tasks. This provides a graphical means to view the interrelationships which exist between the component parts of a program at any desired level.

Even though SM provides all of these very desirable tools, the syntax for logic representation in SM does not seem to mesh well with the simulation modeling environment. There are two reasons for this.

First, the tools are designed for the representation of mathematical models. They seem tailored for use by people with a strong math background. The personnel who do simulation modeling may not have this strong math background and may find it difficult to use these tools. Admittedly, training could eventually reduce this problem.

Second, these mathematical programming tools do not seem to have all of the features required to comfortably represent the simulation logic. One obvious problem is that the current function statements can only be used to generate a number or boolean value. There are many cases where it is necessary to perform a procedural task to generate this number or value, such as the manipulation of paired units in the ONEC model, but SM does not seem to handle this well.

There were many problems directly attributable to the immaturity of the SM concept. These include the problems with index manipulation, construction of model structures, inheritance and the use of attributes. While these problems were very real during our attempt at modeling ONEC, we feel that they are the result of our lack of understanding of the SM system and that they would be overcome with continued exposure.

The bottom line is that in its current form the problems outweigh the benefits for applying SM to discrete event modeling. However, because these benefits are so important we strongly recommend methods be examined to alleviate these problems and make SM more palatable to the simulation domain. Some specific recommendations follow.

B. RECOMMENDATIONS

In the course of this thesis we have developed some thoughts on actions which could be taken to improve the applicability of SM to the discrete event simulation domain. Should these things come to pass, it will be necessary to reassess the

applicability of the new SM to discrete event simulation. This task will be much simpler as the SM tools will be better documented, more familiar and more suited to the task.

1. **Recommendation 1**

A tutorial guide to structured modeling must be provided if SM is ever to mature. SM is far too large to be championed by a single individual. A base of SM practitioners is required to flesh out the SM framework and generate a valid SM environment. This will only happen after the documentation is created which makes the SM tools available to future users. This is provided as a note rather than a recommendation as we understand that Geoffrion is currently working on such a document.

2. **Recommendation 2**

A repository of modeling structures for common modeling requirements should be created. Geoffrion's work on hierarchies, Reference 13, and the sections on modeling tables and inheritance from Chapter IV of this thesis are examples of information which should be placed in this repository as part of the tutorial guide.

3. **Recommendation 3**

The issue of using high order languages as the syntax for the index set statements and generic rules must be examined. Mr Chari is investigating the index set statement issue but we are unaware of anyone examining the generic rule section. Using HOL's in these situations would be a significant improvement for simulation modeling, both by providing the modeler a language he was more familiar with and one which was more in line with the requirements of simulation models.

4. **Recommendation 4**

The impact on the elemental detail tables of incorporating time into the model must be examined. Geoffrion's suggested approach [Ref. 1: pg. 2-91], would generate a model and elemental detail table structure which would work; however the size of the resulting data sets seems to make this a questionable area. The proposed concept of tailored data sets would require a change to SM but would also seem to eliminate some of this size problem. The reduced size of the elemental detail tables would improve the run time of the solver, reduce the demand on data storage and simplify the analysis of the fully evaluated model. This is an essential capability if SM is to be used in the efficient execution of simulation models.

APPENDIX A
ONEC GENERIC STRUCTURE

This Appendix contains the generic structure of the ONEC model in both the genus paragraph and corresponding graphical format. Where possible the genus paragraphs are complete and follow the SM syntax. Areas which could not be completed, due to a lack of information in the ONEC documentation or an inability to correctly use the SM tools, are shown with question marks. In some cases the generic rule sections of the function and test elements are written in a pseudo code manner. This is not correct SM syntax. It is just intended to show the logic which should be in the rule section. In addition, there are explanatory notes throughout the genus paragraphs. The notes are set off with a '*'. Again, this is not SM syntax and serves only to highlight certain aspects of the model.

1. GENERIC STRUCTURE TEXT

IBL /pe/ 1 There is a line called the International Boundary Line. It separates the friendly side of the battlefield from the enemy side.

LOC_IBL (IBL) /a/ {IBL} : 0 < = Y < = 135 There is an IBL on the map. It is described as a straight line and can be represented by a Y Coordinate.

GRID_CELLg /pe/ Size GRID_CELL = 1610 1610 GRID CELLS, each measuring 1km X 3km, are placed on a 35Km X 138km Battlefield with their long sides parallel to the long side of the Battlefield.

*Note how the index set statement shows the maximum number of grid cells to be 1610.

GRID_RELIEF (GRID_CELLg) /a/ {GRID_CELL} : "5Dd, 5Dc, 5Ec, 5Fc" Each GRID_CELL has a relief as indicated by four possible configurations of the NATICK LANDFORM CLASSIFICATION CODE.

*This is an example of an externally indexed genus where the index for the attribute GRID_RELIEF comes from the primitive entity GRID_CELL. So when GRID_RELIEF is referenced in the future it will look like GRID_RELIEFg.

GRID_VEG(GRID_CELLg) /a/ {GRID_CELL} : 0 < = INT < = 10 Each GRID_CELL has a value associated with it that tells the fraction of the cell covered by vegetation.

ROADS_AXIAL(GRID_CELLg) /a/ {GRID_CELL} : "none, primary, secondary, both" Each GRID_CELL has a value for roads in the axial direction.

ROADS_LATERAL(GRID_CELLg) /a/ {GRID_CELL} : Range(ROADS_AXIAL) Each GRID_CELL has a value for roads in the lateral direction.

LOC_GRID_CELL(GRID_CELLg) /a/ {GRID_CELL} : (0 < = X < = 135, 0 < = Y < = 135) The location of each grid cell is shown as an ordered pair of (X,Y) coordinate pairs. The first pair represents the NE corner of the unit. The second pair represents the SW corner of the unit.

RELIEFr/pe/ There is a list of all relief values.

VEGv/pe/ There is a list of all vegetation values.

LARGE_UNITu /pe/ There are many LARGE UNITS to be considered in this model.

LOC_LARGE_UNIT(LARGE_UNITu) /va/ {LARGE_UNIT} : Range (LOC_GRID_CELL) The location of each large unit is shown as a pair of (X,Y) coordinates. The first pair represents the NE corner of the unit. The second pair represents the SW corner of the unit.

LARGE_UNIT_TYPE(LARGE_UNITu) /a/ {LARGE_UNIT} : (List all unit types) Every UNIT has a description which fully defines that UNIT in the Army hierarchy. This will include the LEVEL of the UNIT (Division, Regiment, Battalion, Battery, ???) the ECHELON of the UNIT (First, Second, Reserve) and the TYPE of the UNIT (Artillery or Maneuver).

*This should probably be broken down into an attribute for UNIT_LEVEL, UNIT_TYPE and UNIT_ECHELON.

COMMITTED(LARGE_UNITu) /va/ {LARGE_UNIT} : Logical Need work here. This will show if a SECOND ECHELON UNIT has been COMMITTED for the CALC-DIRECTION Function. But where does info come from?

MOTION(LARGE_UNITu) /va/ {LARGE_UNIT} : Logical This will show for each UNIT if it is already moving. But where does info come from?

ENGAGED(LARGE_UNITu) /va/ {LARGE_UNIT} : Logical This will show if a UNIT is currently engaged in a fire fight. INFO???

INFIGHT(LARGE_UNITu) /va/ {LARGE_UNIT} :(Yes or No??? Portion?) This will show the part of the UNIT ENGAGED in a fire fight. INFO??? How should this be done?

&PAIRED_UNITS This is shown as a module because we were unable to correctly model this area. The genus paragraphs developed in the attempt are shown below.

DIST_RAB_RMB1ER(LOC_LARGE_UNITu1, LOC_LARGE_UNITu2)/f/ Select{LARGE_UNIT} X {LARGE_UNIT}
: *a* abs {[(Y1u1 + Y2u1) / 2 - (Y2u2 + Y1u2) / 2]}
The distance between each Red Artillery Battalion (RAB), index u1, and every Reserve Red Maneuver Batallion of the 1st Echelon (RMB1ER), index u2. The distance is only concerned with the north south separation and is measured from the midpoint of each unit.

*Note the use of the cartesian product in the index set statement. Assuming, for the sake of illustration, that there are 5 RAB and 5 RMB1ER this should generate a 3 column data field with 25 rows. The columns would be for the RAB, RMB1ER and the calculated distance. The rows would be for the 25 possible combinations of the 5 RAB and 5 RMB1ER.

The use of LOC_LARGE_UNIT twice in the generic calling sequence seems a little unusual but it is the only obvious way to introduce two index sets for the same genus in the same function. See Reference 4 page 8 and Reference 1 page 2-94 for similar examples.

It seems to be up to the user to keep track of the indices associated with the RAB and RMB1ER so that the elemental detail tables can be built.

MIN_DIST(DIST_RAB_RMB1ERu1u2)/f/ Select{DIST_RAB_RMB1ER}
: *a* and [*a* min (DIST_RAB_RMB1ERu1,), ord(u2)] This should generate a 5 X 3 data set. The 3 columns would be the RAB, RMB1ER and the specific index of the RMB1ER in the LARGE_UNIT elemental detail table. The 5 rows would be for the 5 RAB.

*This syntax is probably not right, but it should be possible to do what is described. The u1 index is intended to mean process each RAB against every RMB1ER. This is similar to processing an array.

MOVING_MIN(MIN_DISTu1u2, MOTIONu2)/t/ {MIN_DIST}

: @if(MOTIONu2 = TRUE), true, false) If the RMBIER unit paired with the RAB unit is moving then MOVING_MIN is true. This calculation is done for each RAB.

*Passing the index for the RMBIER is unclear. The resulting data set should be the same 5 X 3 data set from MIN_DIST with a T F flag in place of the distance value in the third column.

ORDERS(LARGE_UNITu) /ce/ {LARGE_UNIT} Each LARGE_UNIT has a single set of ORDERS at any specific time.

DESTINATION(ORDERSu) /va/ {LARGE_UNIT} : **Range(LOC_GRID_CELL)** Each set of ORDERS includes a DESTINATION. This destination is expressed as an ordered pair of (X,Y) coordinates.

MISSION(ORDERSu) /va/ {LARGE_UNIT} : **"attack, holding attack, be prepared to attack, delay, withdraw, reserve, move to reinforce, defend fortified position, defend hasty defense, defend prepared position"** Each set of ORDERS includes a MISSION.

MISSION_CHANGE(MISSIONu1)/t/ {LARGE_UNIT):
if MISSIONut < > MISSIONut-1 then TRUE. If UNIT has received a new MISSION since the last time slice then true.

*This shows the problems associated with time dependence. The index "t" stands for time. This would have been used throughout the model had the modeling effort progressed that far. It is just left in here as an example. It will be ingored everyplace else.

GIVEN_ORDERS(ORDERSu)/t/{LARGE_UNIT):
if ORDERSut < > ORDERSu(t-1) then TRUE.

SPEED_FAC_TABLE(RELIEFr, VEGv)/a/ {RELIEF} X {VEG} There is a speed factor for every combination of relief and vegetation.

SPEED_FAC_CELL(GRID_RELIEFg, GRID_VEGg, SPEED_FAC_TABLErv) /f/
: Select {SPEED_FAC_TABLE} Where VEGv = GRID_VEGg and RELIEFr =
Where VEGg = GRID_VEGg and RELIEFr = GRID_RELIEFg

SPEED_FAC_AXIAL(ROADS_AXIALg) /f/ {GRID_CELL} : **RULE???** Each GRID_CELL has a maximum speed factor in the AXIAL direction due to the types of roads present. This is a table look-up and generates a fraction of speed allowed factor. (Pg. 5-9, Table 5-3)

SPEED_FAC_LATERAL(ROADS_LATERALg) /f/{GRID_CELL} : **RULE???** Each GRID_CELL has a maximum speed factor in the LATERAL direction due to the types of roads present. This is a table look-up and generates a fraction of speed allowed factor. (Pg. 5-9, Table 5-3)

ROAD_SPEED_FAC(SPEED_FAC_AXIALg1(u), SPEED_FAC_LATERALg1(u), DIRECTIONu)/f/ {LARGE_UNIT}
: ROAD_SPEED_FACu =
{- SPEED_FAC_AXIALg1 * @abs (@cos DIRECTIONu)- +
- SPEED_FAC_LATERALg1 * @abs (@sin DIRECTIONu)-} /
[@abs (@cos DIRECTIONu) + @abs (@sin DIRECTIONu)]
This combines the speed factors in the AXIAL and LATERAL directions and the large unit DIRECTION of travel into one road speed factor for each large unit.

*This is an interesting case because the indices must define a set of grid cells and units with the same location. The given index set statement shows that this will be done for each large unit but not necessarily for every grid cell. It is not clear who must do the calculations to determine which grid cells are called upon. This looks like a case for the **functional multi-valued dependence index replacement option.** [Ref. 1: pg.2-41] This approach would place the logic of choosing the correct grid cells in the elemental detail tables. This issue was never resolved and the approach shown here would not work. Somewhere the logic to perform this selection of the grid cells must be documented and available for the computer implementation.

It is not always obvious what the resulting index for a genus should be. In the case of ROAD_SPEED_FAC it looks like it could be "gu". However, it is actually just

u In these cases the index set statement provides the clue. Notice that the index set statement defines the size of the elemental detail table to be the same as LARGE_UNIT. This means that the index "u" will be all that is required to provide an unambiguous key.

COM_SPEED_FAC_CELL(SPEED_FAC_CELLgl(u), ROAD_SPEED_FAC(u)) (LARGE_UNIT) = COM_SPEED_FAC_CELLgl = SPEED_FAC_CELLgl * ROAD_SPEED_FACu
This combines the speed factors created by TERRAIN, VEGETATION, ROADS and unit DIRECTION into one factor for each large unit.

*Note the same issues mentioned above for ROAD_SPEED_FAC also apply here.

WEAPONw /pe/ There are many Weapons in this model. This approach assumes that weapons are accounted for by groups in weapon types, not as individual units.

WEAPON_TYPE(WEAPONw)/a/{WEAPON} : (A list of weapon types goes here.) There are many TYPES of WEAPON.

WEAPON_RANGE(WEAPONw)/a/{WEAPON} : (A list of all weapon ranges) Each WEAPON_TYPE has a RANGE.

WEAPON_LIST(WEAPONw, LARGE_UNITu)/ce/
Select {WEAPON} X {LARGE_UNIT}
Where w covers {LARGE_UNIT}
Each LARGE_UNIT has a list of all WEAPONS associated with that UNIT.

%AVAIL_WEAPON(WEAPON_LISTwu)/va/{WEAPON_LIST} : 0 < = Int < = 100 There is an accounting for each WEAPON_TYPE in a UNIT. This shows the % of that WEAPON_TYPE that is still active.

%AMMO_WEAPON(WEAPON_LISTwu)/va/{WEAPON_LIST} : Range(%AVAIL_WEAPON) There is an accounting for the AMMO for each WEAPON_TYPE in a UNIT. This is shown as a % of the AMMO left for that WEAPON_TYPE.

INFIGHT_WEAPON(WEAPON_LISTwu)/va/{WEAPON_LIST} : Range(LOC_GRID_CELL) Only certain weapons in a UNIT will be in the fire fight geometry. This would be a relationship between the geometry of the firefight and the weapon distribution. It is not clear if this should be a % or a geographical area. For illustration it is shown as an area.

SMALL_UNITs /pe/ There are many Small Units to be considered in this model. A small unit is one associated with a large unit. It gets it's mission speed, and direction from that large unit. Examples are radars, command posts and recon units.

LOC_SMALL_UNIT(SMALL_UNITs) /va/ {SMALL_UNIT} : Range(LOC_GRID_CELL) Every SMALL_UNIT has a location that can be expressed as a pair of (X,Y) coordinates.

SMALL_UNIT_TYPE(SMALL_UNITs) /a/ {SMALL_UNIT} : (List all types) Every UNIT has a description which fully defines that UNIT in the Army hierarchy. This will include the TYPE and ECHELON of the SMALL UNIT. (COMMAND POSTS, RADARS...)

ASS_UNIT(LARGE_UNITu, SMALL_UNITs)/ce/
Select {LARGE_UNIT} X {SMALL_UNIT}
where s covers {LARGE_UNIT}
Each LARGE_UNIT has in it a set of SMALL_UNITs.

TARGET_LIST(ASS_UNITus)/ce/ Select{ASS_UNIT} Each LARGE_UNIT has a list of all Targets associated with that UNIT. This does not account for the case where weapons are targets also.

ALIVE_TARGET(TARGET_LISTus)/va/{TARGET_LIST} : Logical There is an accounting for each TARGET in a UNIT. Assume this is done on a per target basis.

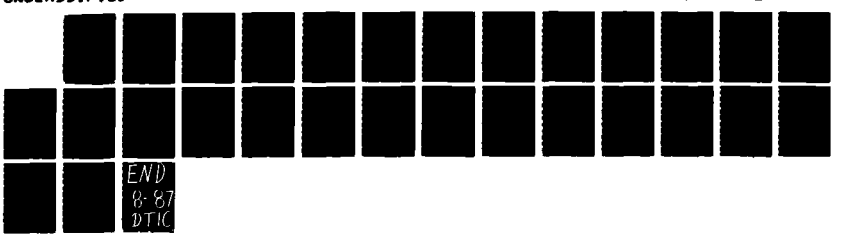
INFIGHT_TARGET(TARGET_LISTus)/va/{TARGET_LIST} : Logical Only certain TARGETs in a UNIT will be in the fire fight geometry.

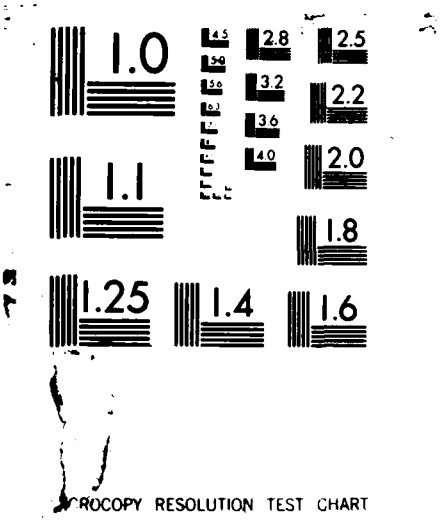
*How should this be calculated? This case is different than the INFIGHT_WEAPON case because the SMALL_UNITS are treated as individual units with specific locations.

```
CALC_DIRECTION(LOC_LARGE_UNITu,                    LARGE_UNIT_TYPEu,
DESTINATIONu,      COMMITTEDu,      GIVEN_ORDERSu,      MOTIONu,
MISSION_CHANGEu, MISSIONu)/t/ {LARGE_UNIT}
 : If GIVEN_ORDERS
 and
    {([LARGE(UNIT_TYPE = BLUE ARTY UNIT any ECHELON)
    or (BLUE CMD POST > BATTALION)
    and (NOT MOVING)
    and (DESTINATION < > LOCATION)]
 or
    [LARGE_UNIT_TYPE = BLUE MANEUVER UNIT any ECHELON)
    and (MISSION_CHANGE)
    and (MISSION < > DEFEND)]
 or
    [LARGE_UNIT_TYPE = RED MANEUVER UNIT 2nd ECHELON
     2nd DIVISION)
    or (LARGE_UNIT_TYPE = RED COMMAND POST > BATTALION
        2nd ECHELON 2nd DIVISION)
    and (COMMITTED)]}
then
    CALC_DIRECTION = true

DIRECTION(LOC_LARGE_UNITu,    LARGE_UNIT_TYPEu,    DESTINATIONu,
CALC-DIRECTIONu)/f/{LARGE_UNIT}
 : if CALC_DIRECTION
 then
    if [LARGE (UNIT_TYPE = RED ARTY UNIT any ECHELON)
       or (LARGE_UNIT_TYPE = RED MANEUVER UNIT 1st ECHELON
           1st DIVISION)]
    then
          DIRECTION = 270 Degrees
    else
          DIRECTION = (Equations 5-1, 5-2, 5-3, 5-4, 5-5, Pg.5-6)

MAX_SPEED_UNIT(LOC_LARGE_UNITu, LOC_IBL) /f/ {LARGE_UNIT}
If LOC_LARGE_UNITu = friendly side of IBL
 then
    MAX_SPEED_UNIT = 25km/Hr
 else
    MAX_SPEED_UNIT = 15km/Hr.

MISSION_REL_FACTOR(COM_SPEED_FAC_CELLu,        LOC_LARGE_UNIT
LARGE_UNIT_TYPEu, MISSIONu)/f/{LARGE_UNIT}:
 : If MISSIONu = DELAY
 then
    MISSION_REL_FACTOR = 0.75
 else
    If (MISSIONu = ATTACK) and
       (TYPE = 1st ECHELON DIVISION)
       then
          Select UNITu * GRID_CELLu
          for LOCATIONu intersect LOC_LARGE_UNIT
          SORT on COMBINED SPEED FAC CELL ascending
          MISSION_REL_FACTOR = slowest(COMBINED SPEED
          FACTOR CELL
       else
          If (MISSIONu = ATTACK) and
             (UNIT_TYPEu = 2nd ECHELON DIVISION
             then
                Select UNITu * GRID_CELLu for
                LOCATIONu intersect LOC_LARGE
                SORT on COMBINED SPEED FAC
```

AD-A181 993 THE APPLICABILITY OF STRUCTURED MODELING TO DISCRETE EVENT SIMULATION SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D J PATRICK MAR 87 2/2

UNCLASSIFIED F/G 15/6 NL




1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.8
ROCOPY RESOLUTION TEST CHART

```
            MISSION_REL_FACTOR = fastest SPEED_FAC_CELL
        else
            MISSION_REL_FACTOR = 1
```

MISSION_REL_FACTORS seems to apply to only the BLUE UNITS DELAYING or the RED UNITS ATTACKING. It requires a sorted list of the COMBINED SPEED FACTORS CELL for each CELL that the RED UNIT is sitting on. This requires a link between the UNIT LOCATION and the GRID_CELL LOCATION.

*The index set statement shows that there will be a result for each unit. This means that the resulting index for MISSION_REL_FACTORS will be a "u". This is because the unit is all that is required to provide an unambigious key value.

```
REL_COMBAT_RATIO_FACTOR(LARGE_UNIT_TYPEu, %AVAIL_WEAPONwu,
INFIGHT_WEAPONwu, ENGAGEDu)/f/{LARGE_UNIT}
 ; If (LARGE_UNIT_TYPEu = BATTALION) and
    (LARGE_UNIT_TYPEu < > RED ARTY)
 then
    If LARGE_UNIT not ENGAGED
       then
          REL_COMBAT_RATIO_FACTOR = 1
       else
          u1 = BLUE UNIT  u2 = RED UNIT
          Select %AVAIL_WEAPONwu1 * INFIGHT_WEAPONwu1
          Count1
          Select %AVAIL_WEAPONwu2 * INFIGHT_WEAPONwu2
          Count2
          REL_COMBAT_RATIO_FACTOR = COUNT2 / COUNT 1
          REL_COMBAT_RATIO_FACTOR = Table look up.
          Page 5-12, Table 5-4

ARTY/CAS_FACTOR(ENGAGEDu,                    %AVAILABLE_WEAPONwu,
INFIGHT_WEAPONwu)/f/ {LARGE_UNIT}
 ; If ENGAGED
 then
    u1 is UNIT under consideration
    u2...ux are UNITS ENGAGED with u1
    Select  INFIGHT_WEAPONu2
          for (WEAPON = CAS) or (WEAPON = ARTY)
    Repeat for all ENGAGING UNITS u2..ux
    Count
    If Count > 0
       then
          ARTY/CAS FACTOR = 0.75
       else
          ARTY/CAS FACTOR = 1.0

ALLOWED_UNIT_SPEED(MAX_SPEED_UNITu,    MISSION_REL_FACTORSu,
REL_COMBAT_RATIO_FACTORu, ARTY/CAS FACTORu)/f/{LARGE_UNIT}
 ;       ALLOWED_UNIT_SPEED       =       (MAX_SPEED_UNIT       *
MISSION_REL_FACTORS * REL_COMBAT_RATIO_FACTOR * ARTY/CAS
FACTOR)

RED_UNIT_INTEGRITY(LOC_LARGE_UNITu,          %AVAIL_WEAPONwu,
LARGE_UNIT_TYPEu,       INFIGHT_WEAPONwu,ASS_UNITus)/f/       Select
{LARGE_UNIT} ; Rule?

ACT_SPEED_UNIT(ALLOWED_UNIT_SPEEDu,     RED_UNIT_INTEGRITYu)/f/
{LARGE_UNIT}
 ; If LARGE_UNIT_TYPE = RED
 then
    ACT_SPEED_UNIT = RED_UNIT_INTEGRITY
 else
    ACT_SPEED_UNIT = ALLOWED_UNIT_SPEED
```

2. GENERIC STRUCTURE GRAPHICAL

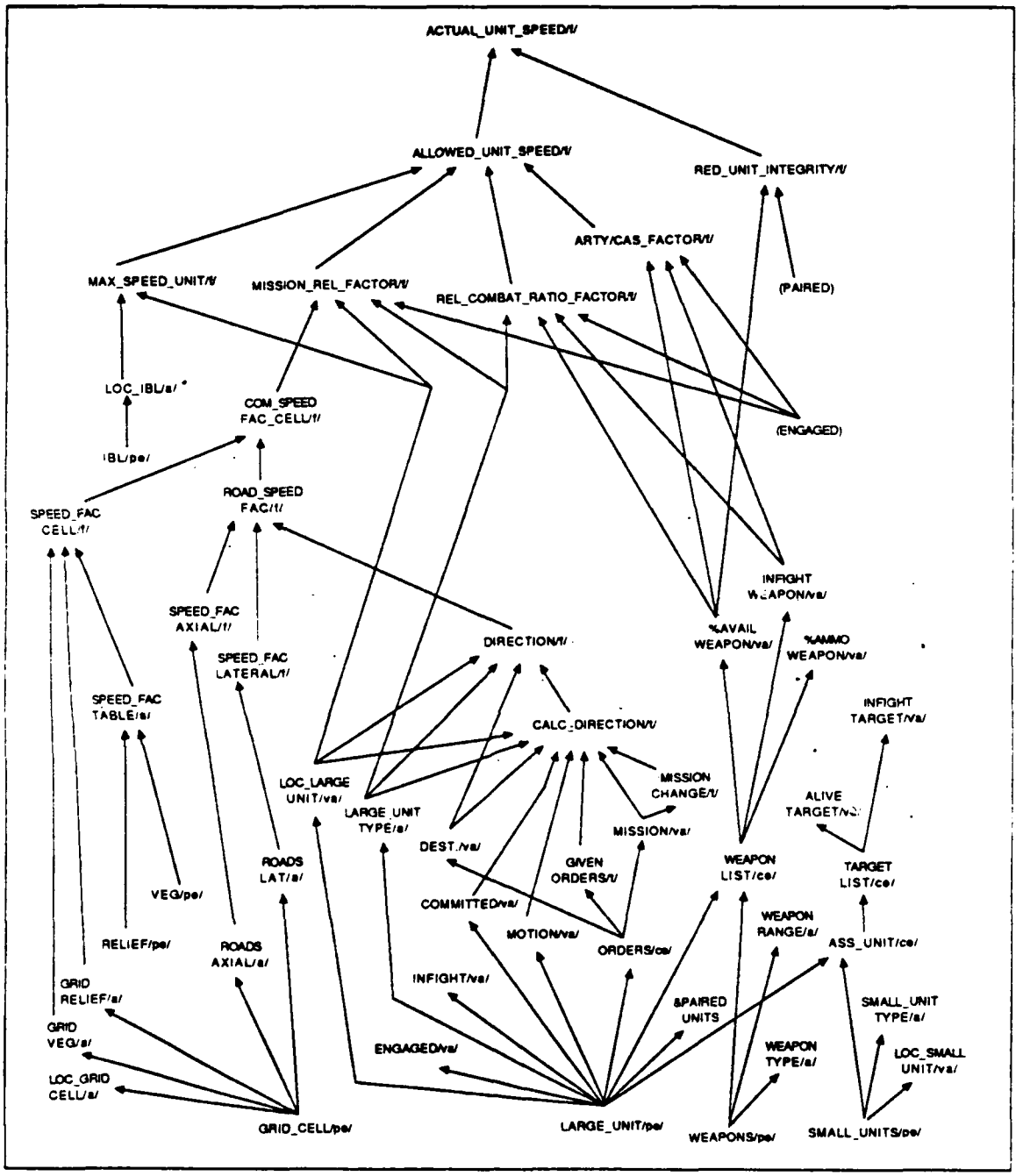


Figure A.1 Generic Structure Graphical.

APPENDIX B
ONEC MODULAR STRUCTURE

This Appendix contains a modular structure, both text and graphical, of the ONEC model. There are numerous modular structures which could be fitted to the generic structure. It is important to remember that this is just one.

The format of the text section is not the same as you would find in Geoffrion's work because the genus paragraph information has been omitted. In an actual model implementation the generic structure is always shown within a modular structure. In this thesis the generic structure was shown in Chapter 4 and Appendix A without the modular structure for illustration purposes. It would serve no useful purpose to repeat the genus paragraph information here. But keep in mind that in a normal presentation the modular structure would be shown with the entire genus paragraph and not just the genus names.

1. MODULAR STRUCTURE TEXT

&ONEC The *ONEC* structured model.

&BATTLEFIELD The *battlefield representation* section.

&IBL The *International Boundary Line* section.

IBL/pe/

LOC_IBL/a/

&GRID The *grid cell* section.

GRID_CELL/pe/

GRID_RELIEF/a/

GRID_VEG/a/

ROADS_AXIAL/a/

ROADS_LATERAL/a/

LOC_GRID_CELL/a/

&UNIT The *large unit* section.

LARGE_UNIT/pe/

LOC_LARGE_UNIT/va/

LARGE_UNIT_TYPE/a/

COMMITTED/va/

MOTION/va/

ENGAGED/va/
INFIGHT/va/
PAIRED_UNITS/va/
&WEAPON The *weapon* section.
WEAPON/pe/
WEAPON_TYPE/a/
WEAPON_RANGE/a/
&SMALL_UNIT The *small unit* section.
SMALL_UNITS/pe/
LOC_SMALL_UNIT/va/
SMALL_UNIT_TYPE/a/
&WEAPON_LIST The *combination* of *weapons* and *large units.*
WEAPON_LIST/ce/
%AVAIL_WEAPON/va/
%AMMO_WEAPON/va/
INFIGHT_WEAPON/va/
&TARGETS The *combination* of *large* and *small units* and *target establishment.*
ASS_UNITS/ce/
TARGET_LIST/ce/
ALIVE_TARGETS/va/
INFIGHT_TARGET/va/
&MOVEMENT *Speed* and *direction* of *large units.*
&MISSION *Orders* for *large units.*
ORDERS/ce/
DESTINATION/va/
MISSION_CHANGE/t/
GIVEN_ORDERS/t/
MISSION/va/
&DIRECTION Which way did he go.
CALC_DIRECTION/t/
DIRECTION/f/
&COM_SPEED_FAC *Speed decrement factors.*
&SPEED_TABLE *Vegetation and Relief speed factor table.*
RELIEF/pe/
VEG/pe/

SPEED_FAC_TABLE/a/
COM_SPEED_FAC_CELL/f/
SPEED_FAC_CELL/f/
ROAD_SPEED_FAC/f/
SPEED_FAC_AXIAL/f/
SPEED_FAC_LATERAL/f/
&MISSION_SPEED Combination of *all speed factors.*
RED_UNIT_INTEGRITY
ACTUAL_UNIT_SPEED/f/
&POSSIBLE_UNIT_SPEED *Max speed* possible for units.
ALLOWED_UNIT_SPEED/f/
MAX_SPEED_UNIT/f/
MISSION_REL_FACTORS/f/
REL_COMBAT_RATIO_FACTOR/f/
ARTY/CAS_FACTOR/f/

2. MODULAR STRUCTURE GRAPHICAL

Figures B.1 through B.5 show the graphical representation of the modular structure just presented. Figure B.1 shows the first three levels of the modular tree, Figures B.2, B.3 and B.4 show the fourth level of the tree and Figure B.5 shows the fifth and last level.

3. MODULAR GRAPHS

This section shows the module graph representation of the modular structure just presented. Chapter 4 presented a step-by-step look at how these modules fit together. This appendix will provide a single big picture figure, Figure B.6, which shows the entire ONEC model in a module graph form. The remaining 14 Figures (Figures B.6 through B.21) provide the detailed graphical representations of each individual module. Keep in mind that if you were to replace all of the modules in the big picture figure with the details from the individual module figures you would end up with the complete generic structure.

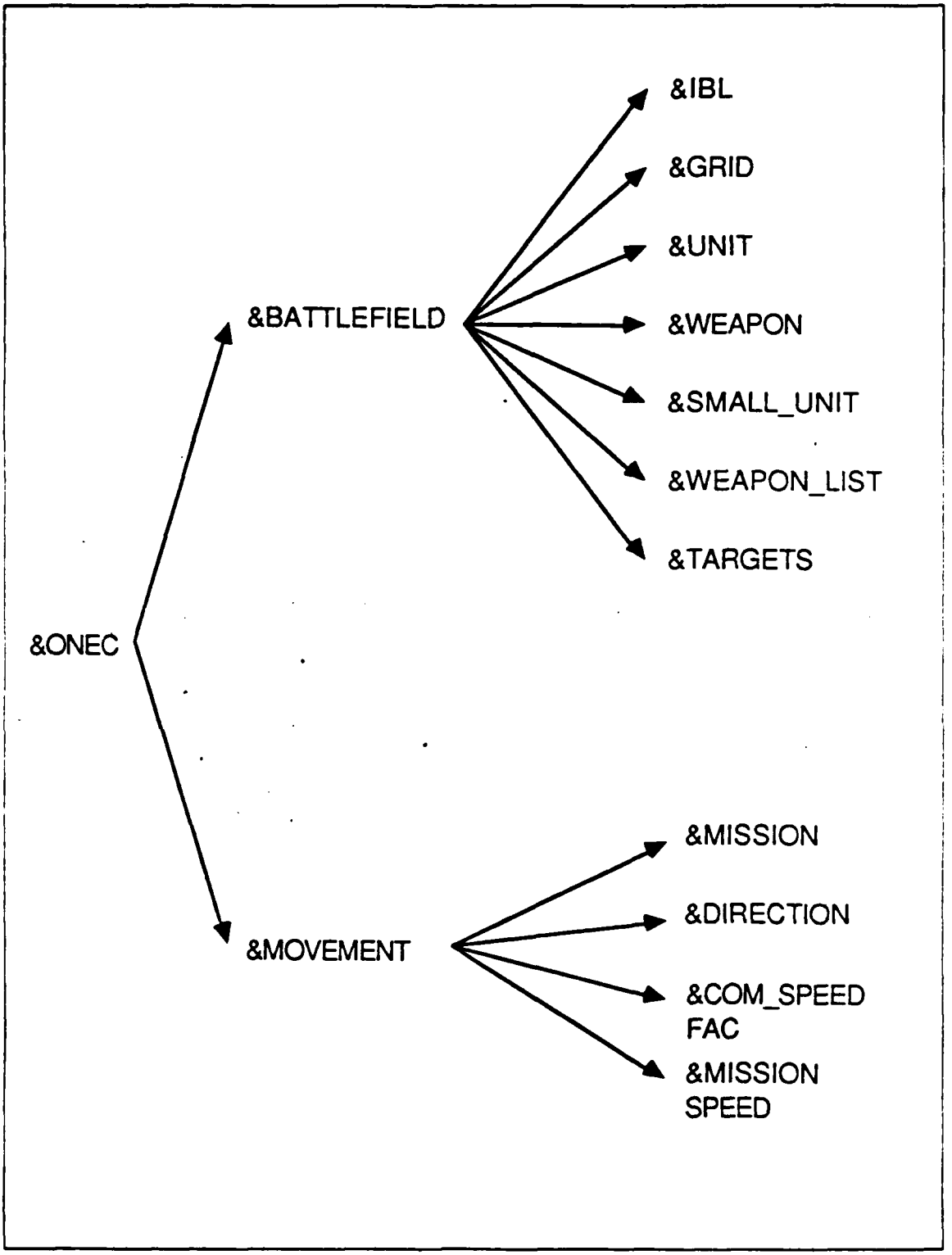


Figure B.1 First Three Levels of Module Structure Tree.

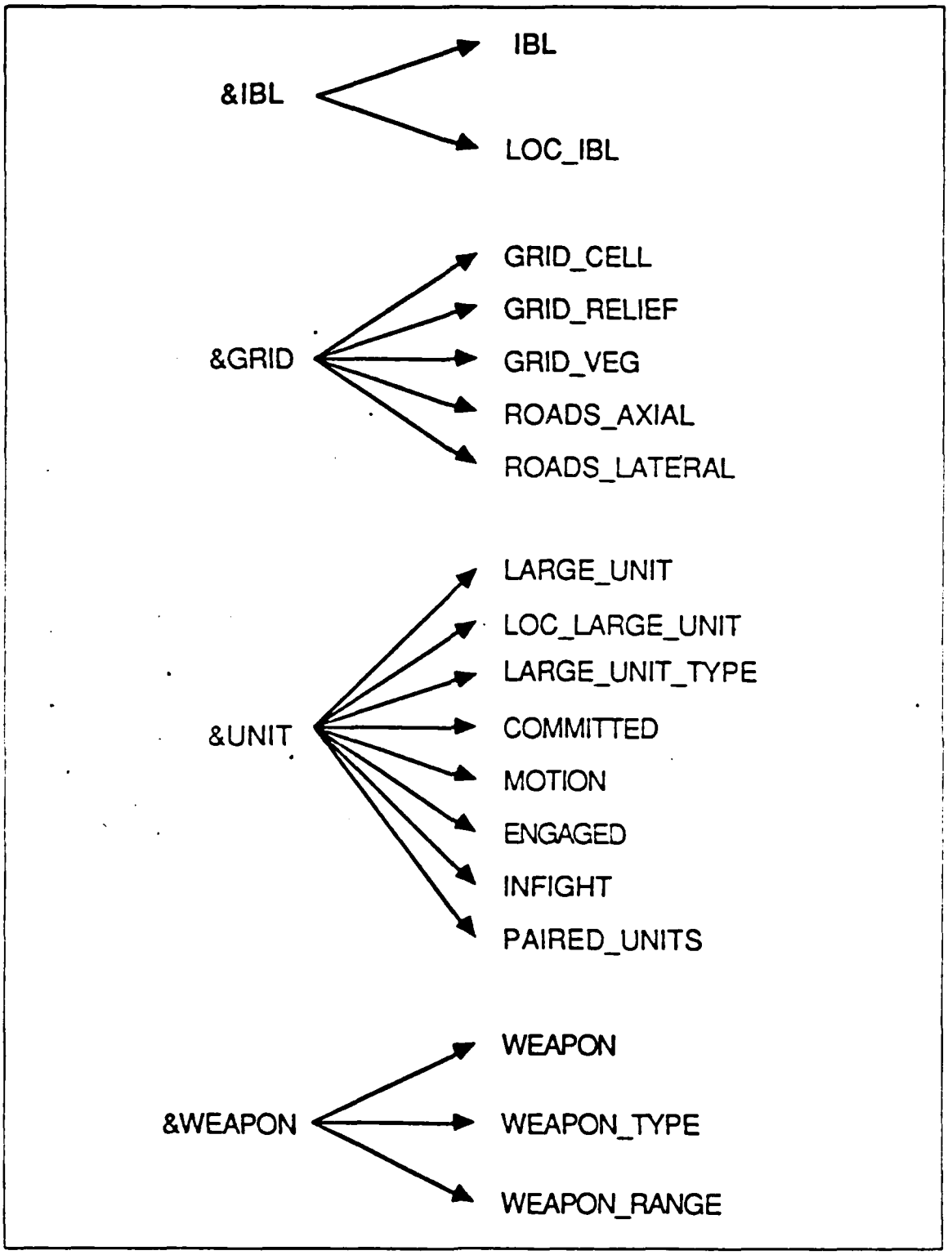


Figure B.2 Fourth Level of Module Structure Tree.

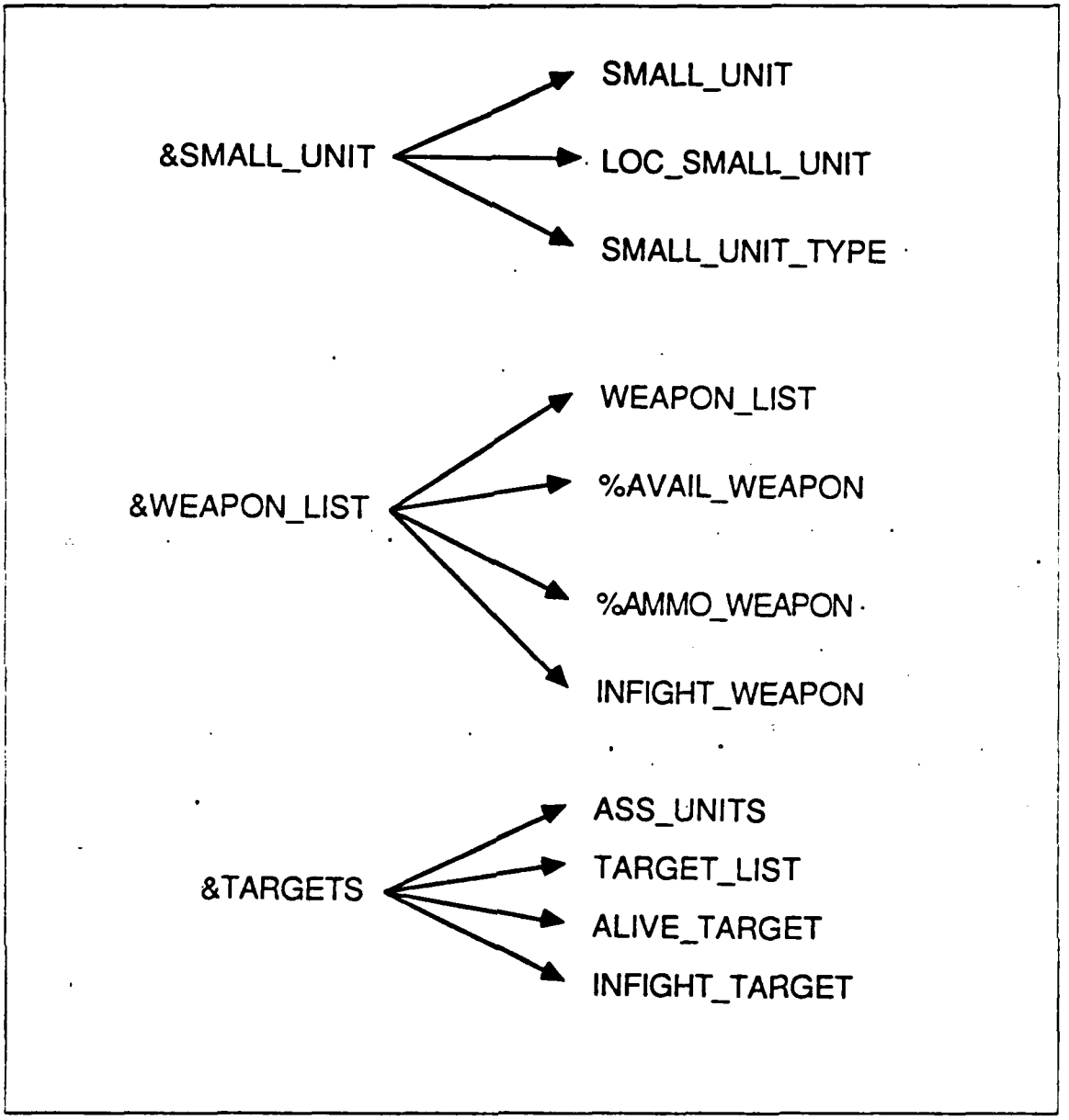


Figure B.3 Fourth Level of Module Structure Tree Continued.

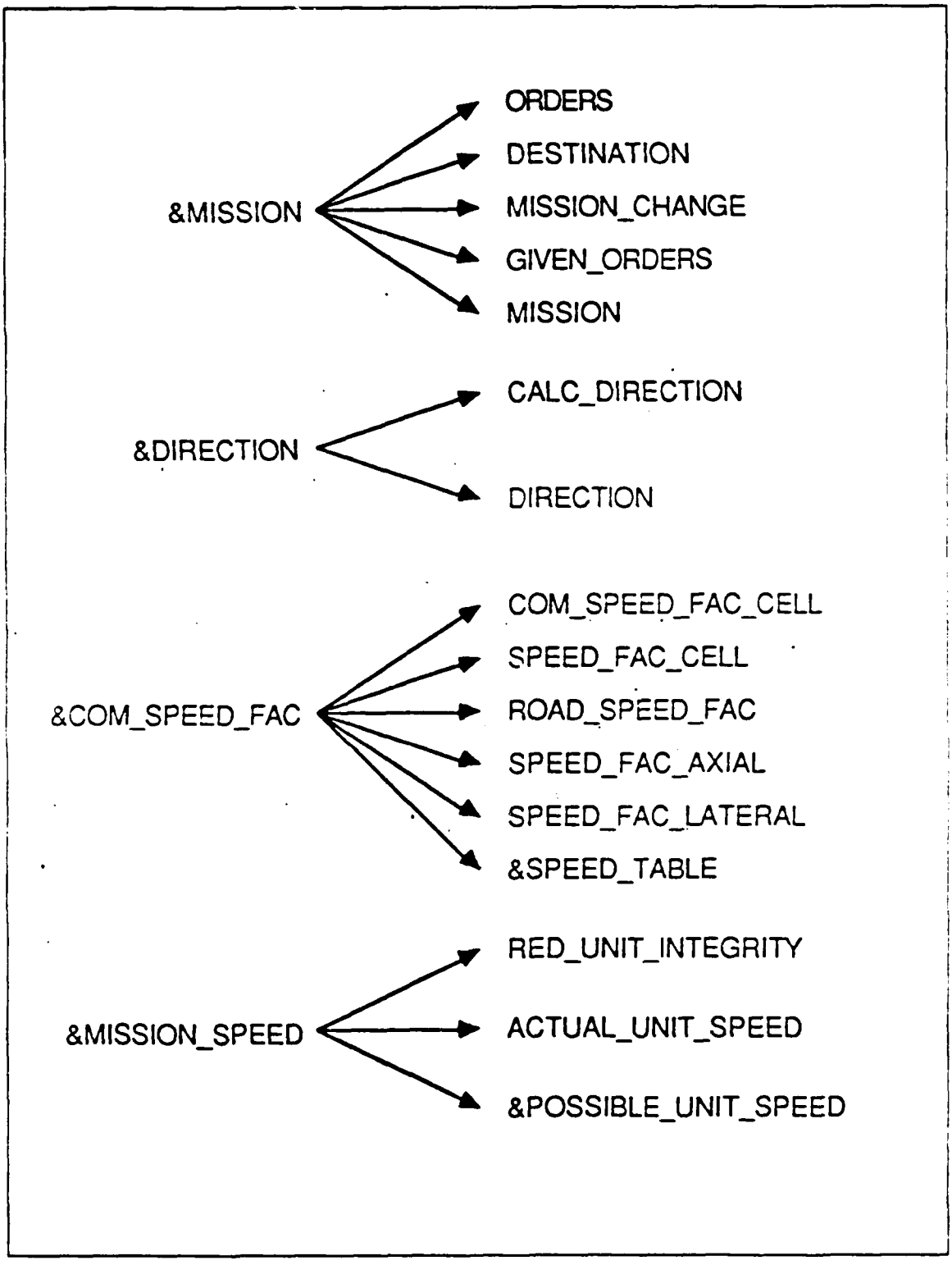


Figure B.4 Fourth Level of Module Structure Tree Continued.

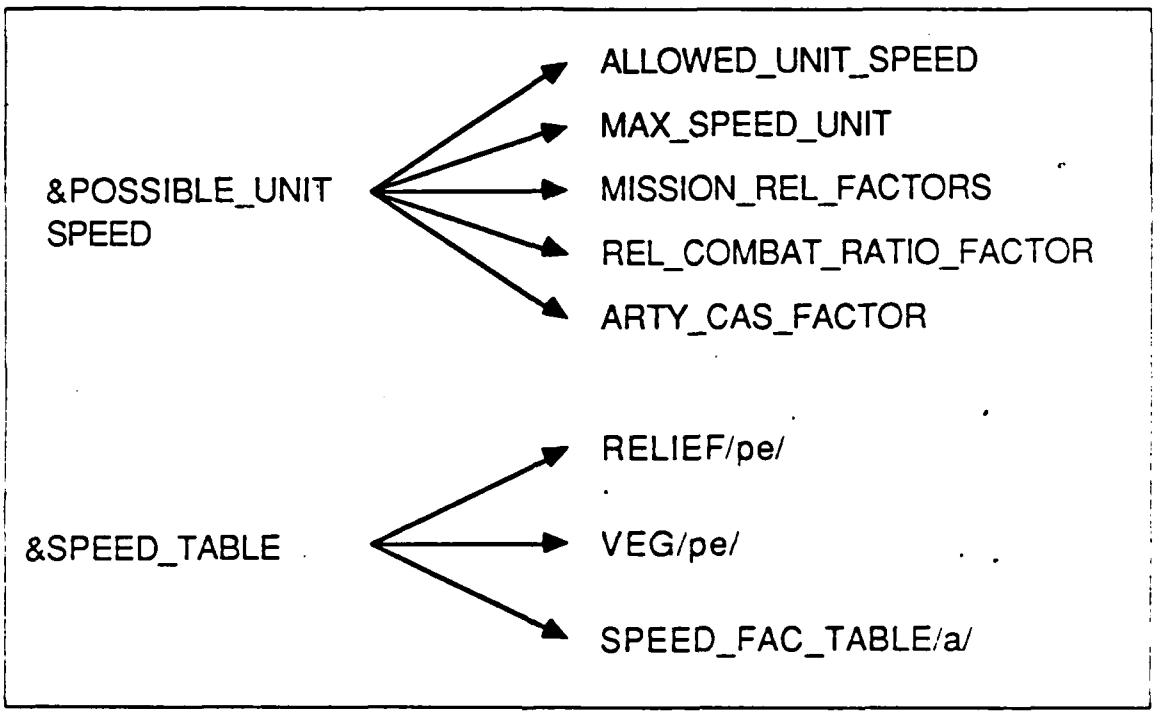


Figure B.5 Fifth Level of Module Structure Tree.

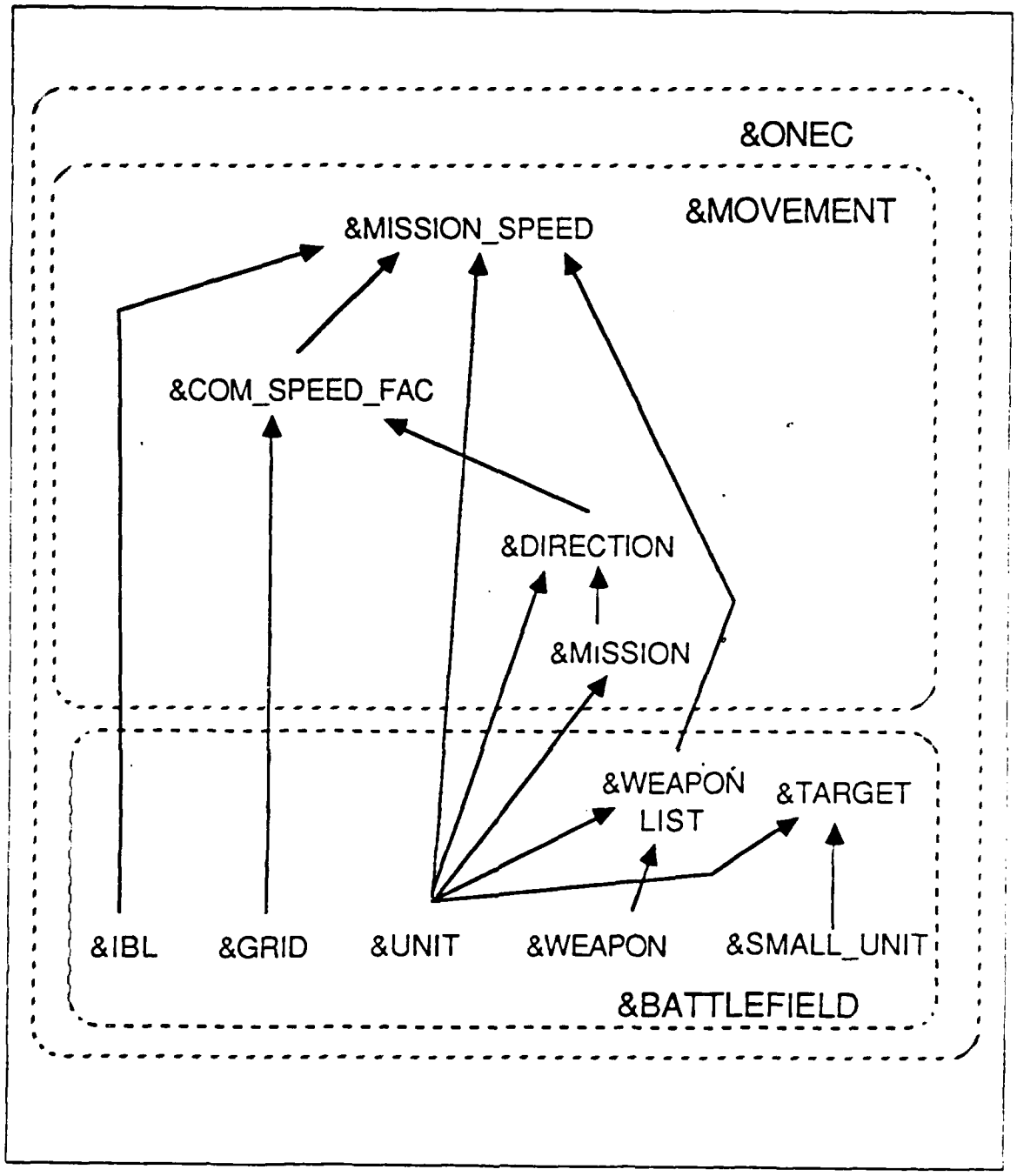


Figure B.6 Module Graph of the ONEC Model.

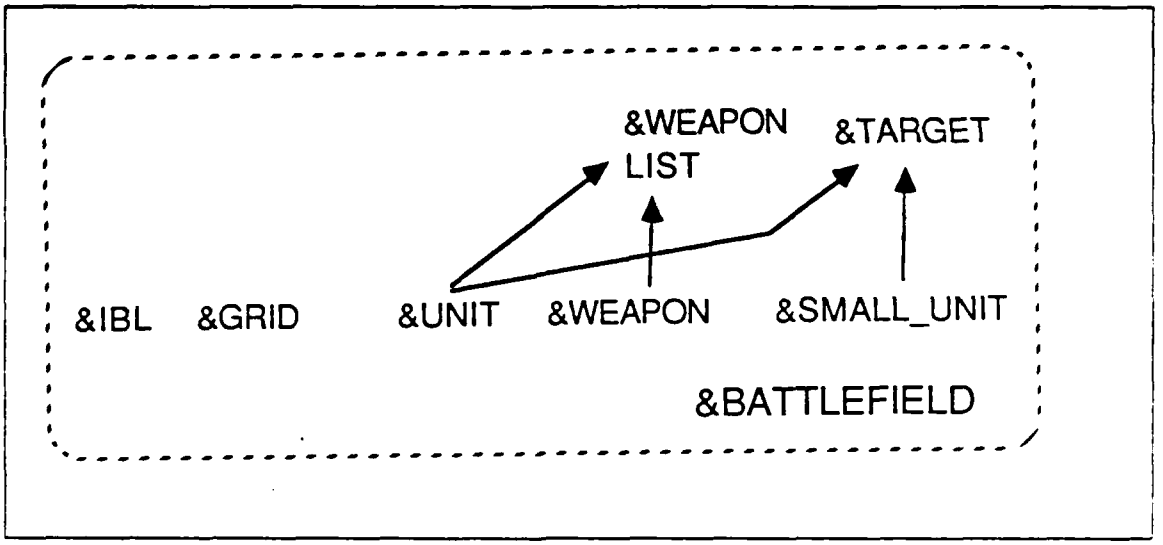


Figure B.7 Module BATTLEFIELD.

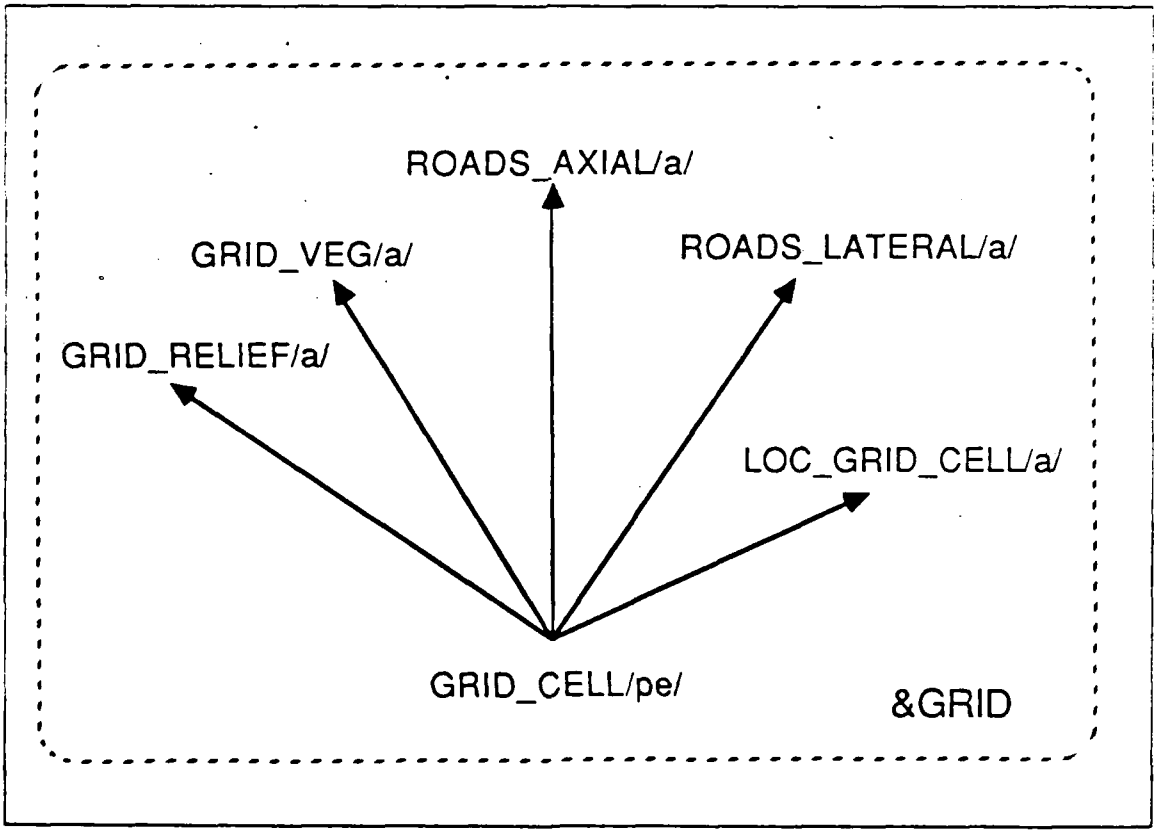


Figure B.8 Module GRID.

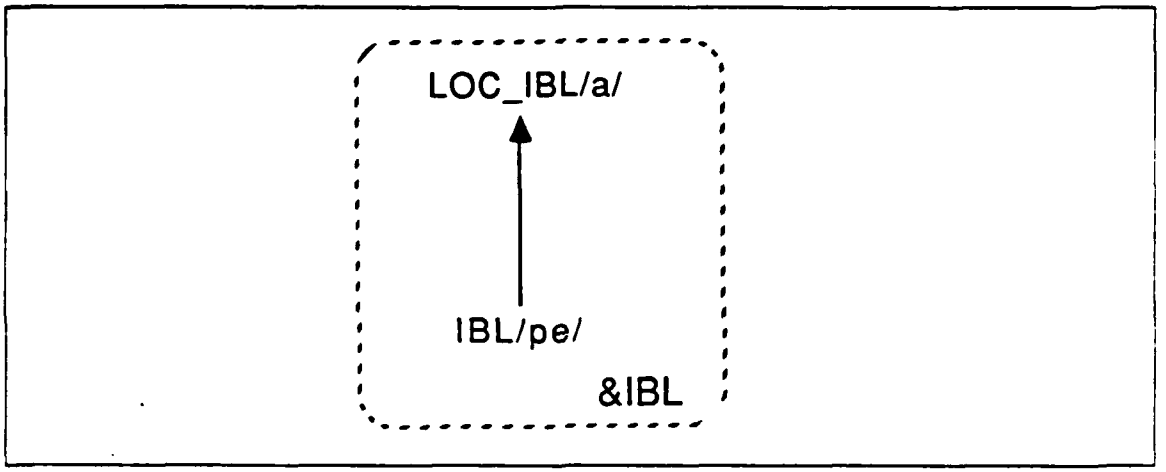


Figure B.9 Module IBL.

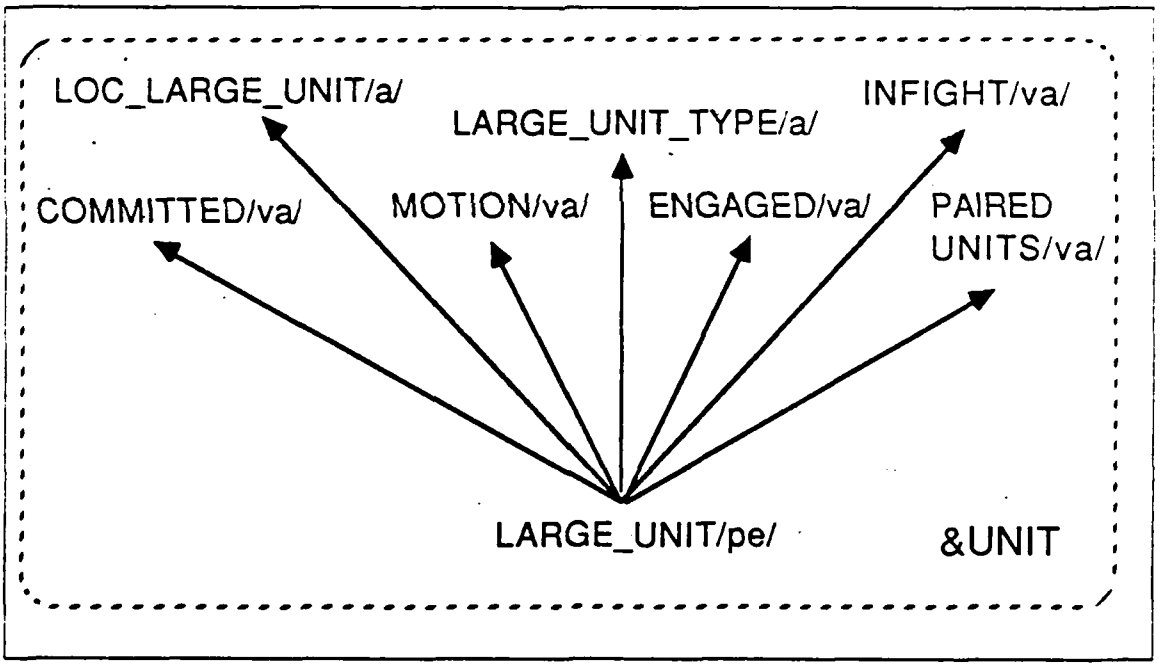


Figure B.10 Module UNIT.

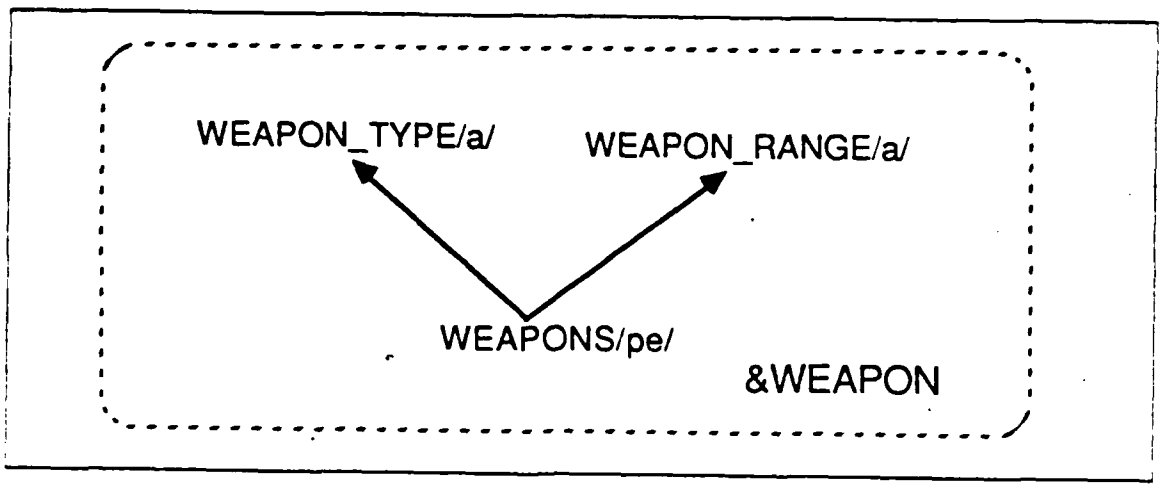


Figure B.11 Module WEAPON.

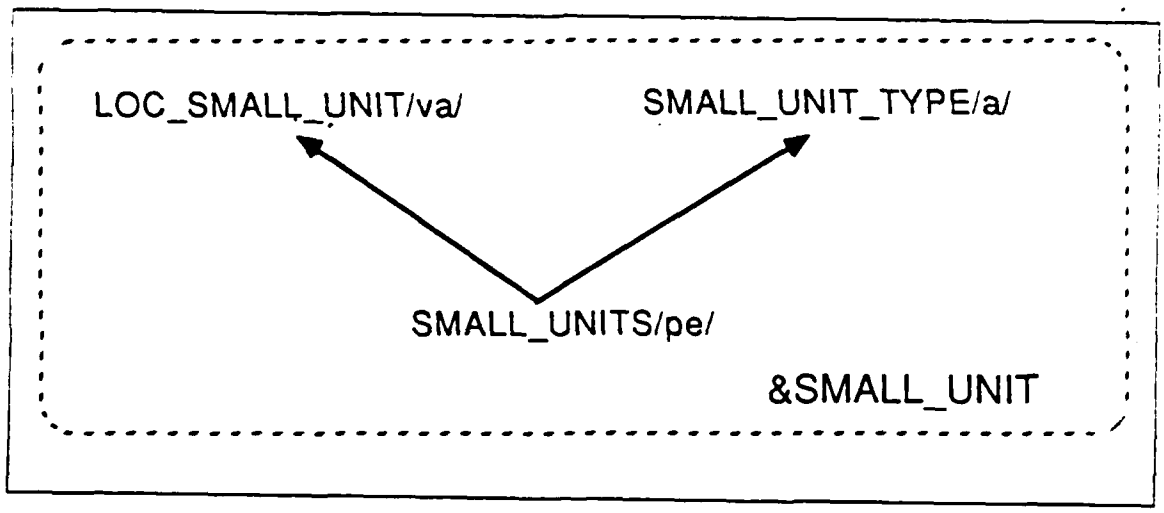


Figure B.12 Module SMALL_UNIT.

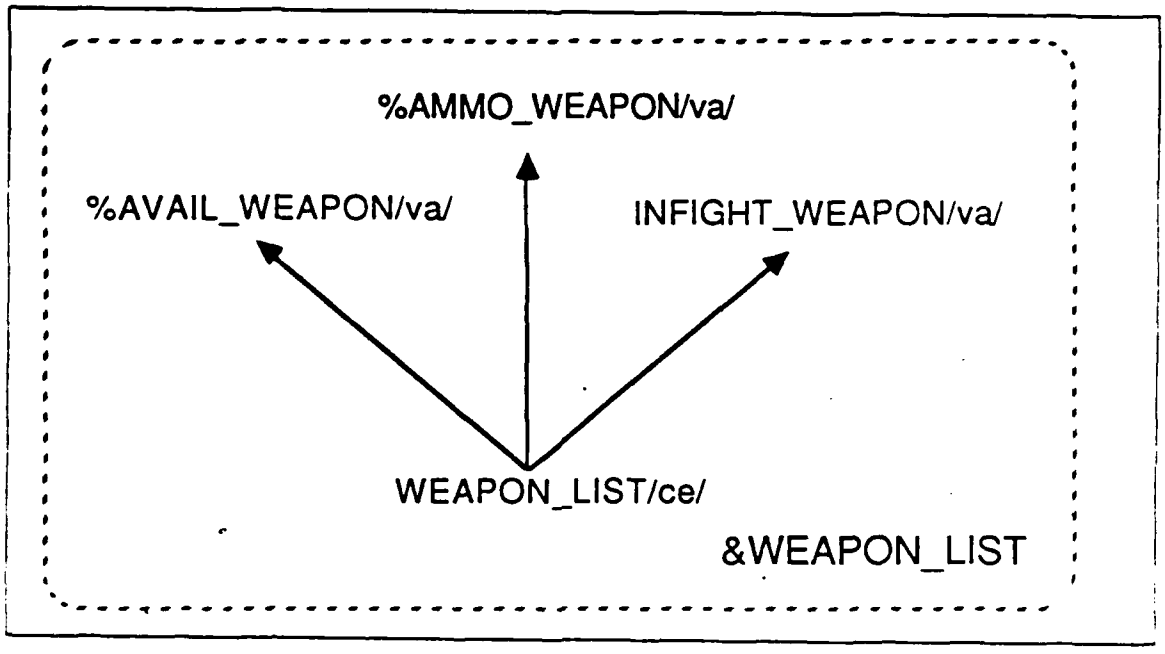


Figure B.13 Module WEAPON_LIST.

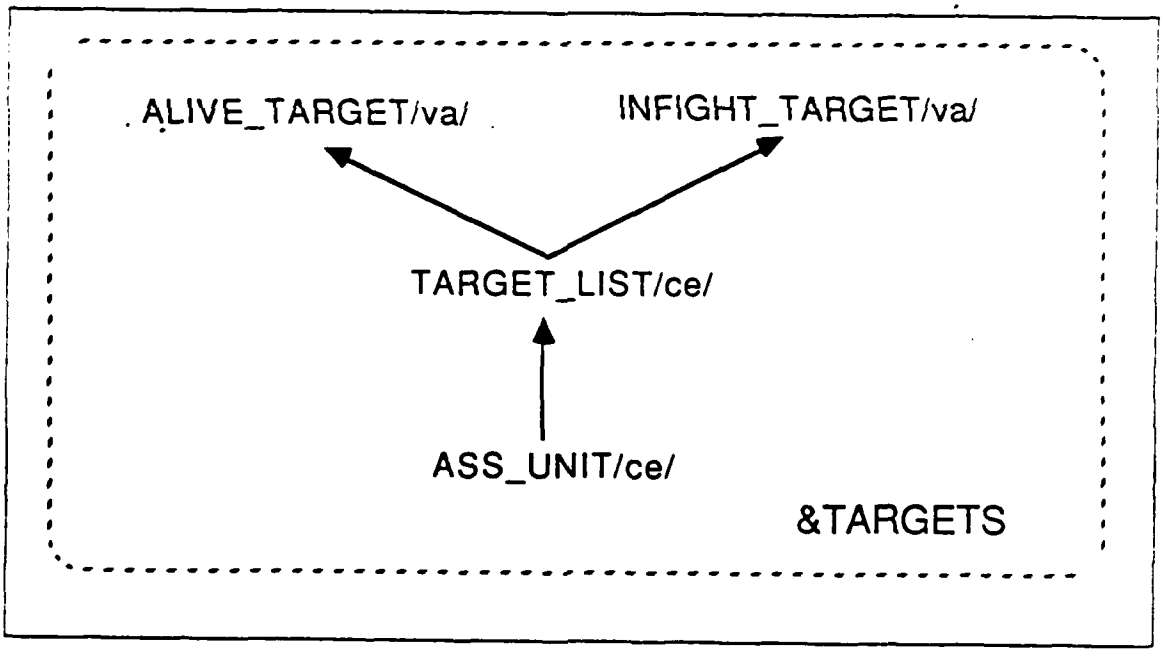


Figure B.14 Module TARGETS.

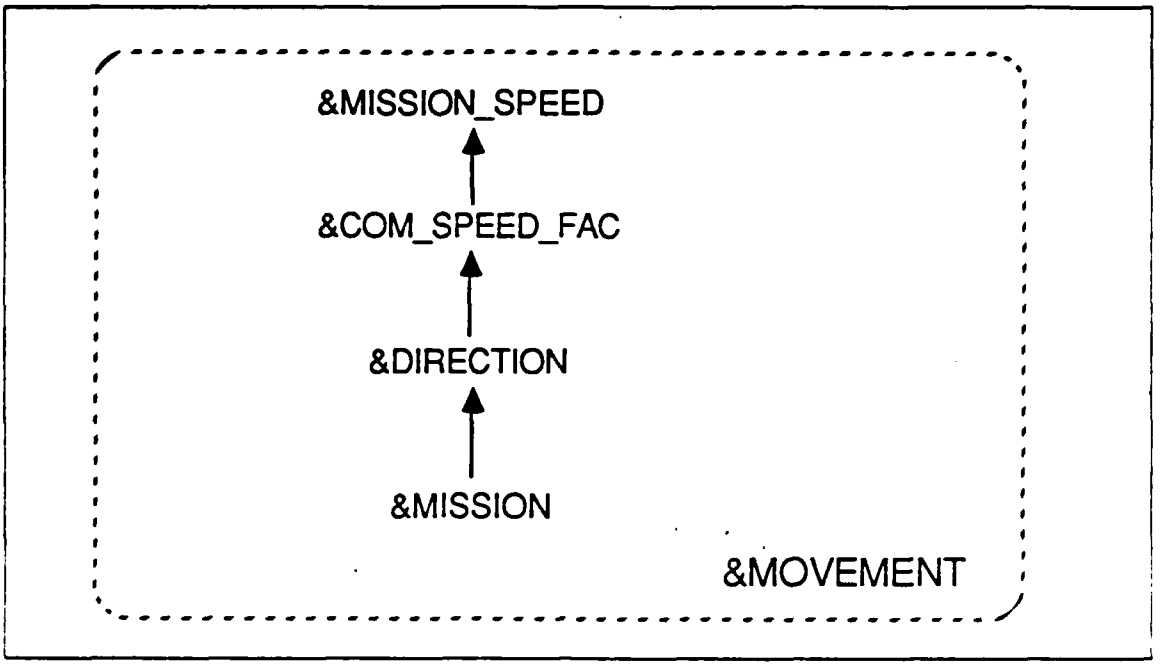


Figure B.15 Module MOVEMENT.

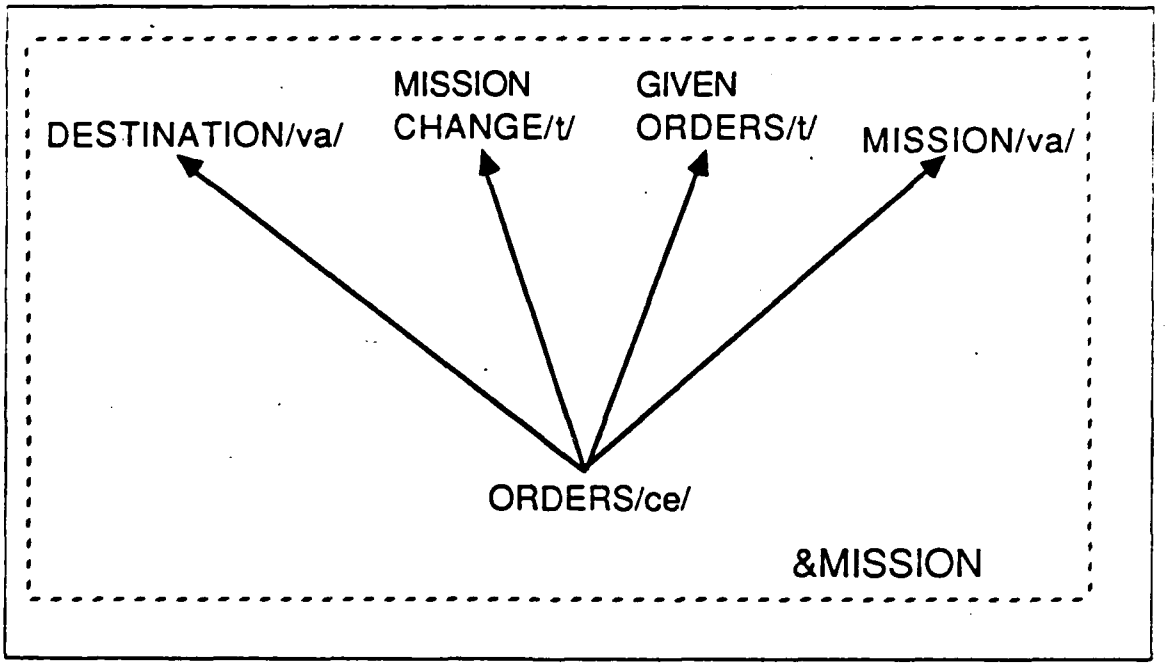


Figure B.16 Module MISSION.

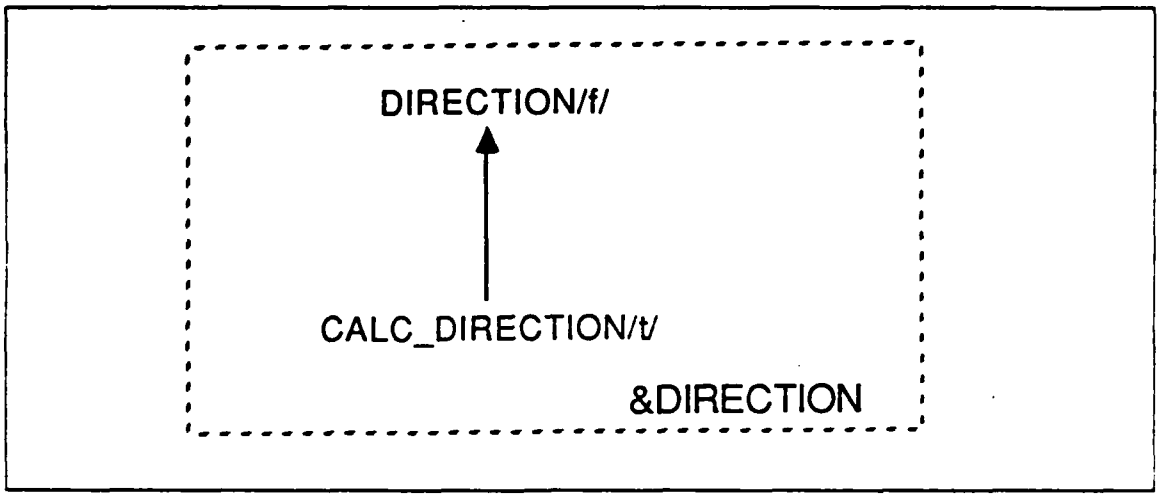


Figure B.17 Module DIRECTION.

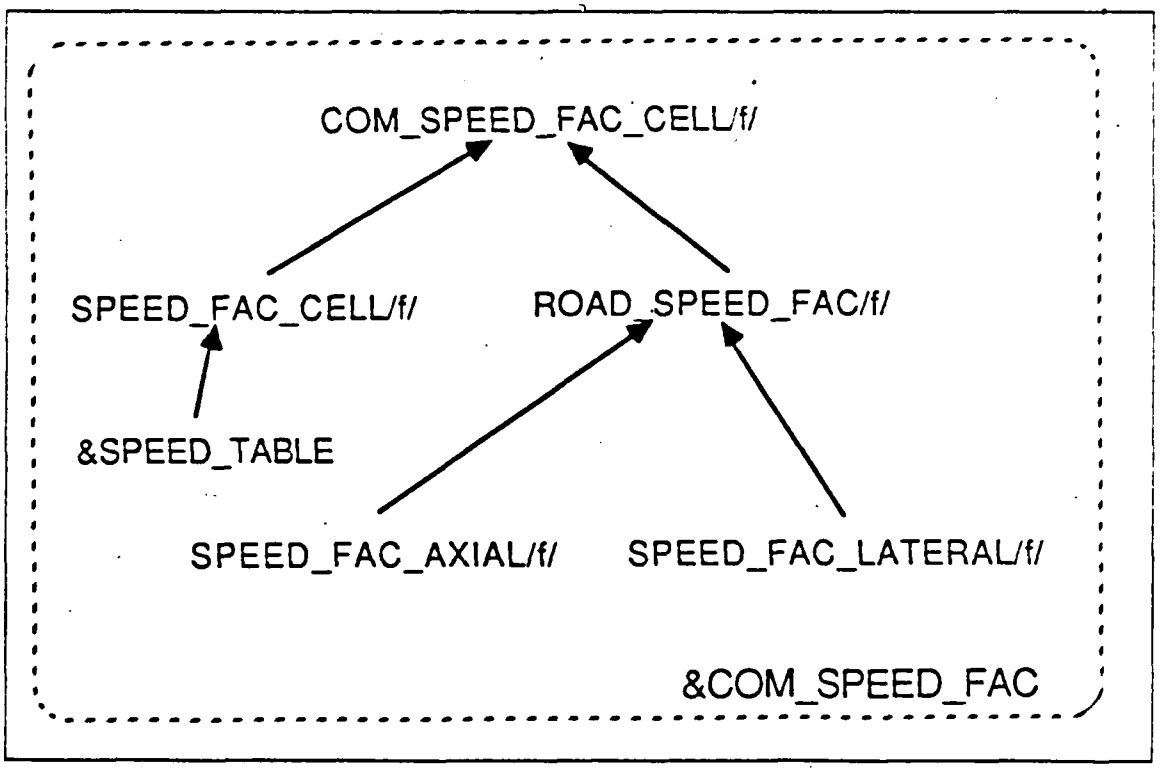


Figure B.18 Module COM_SPEED_FAC.

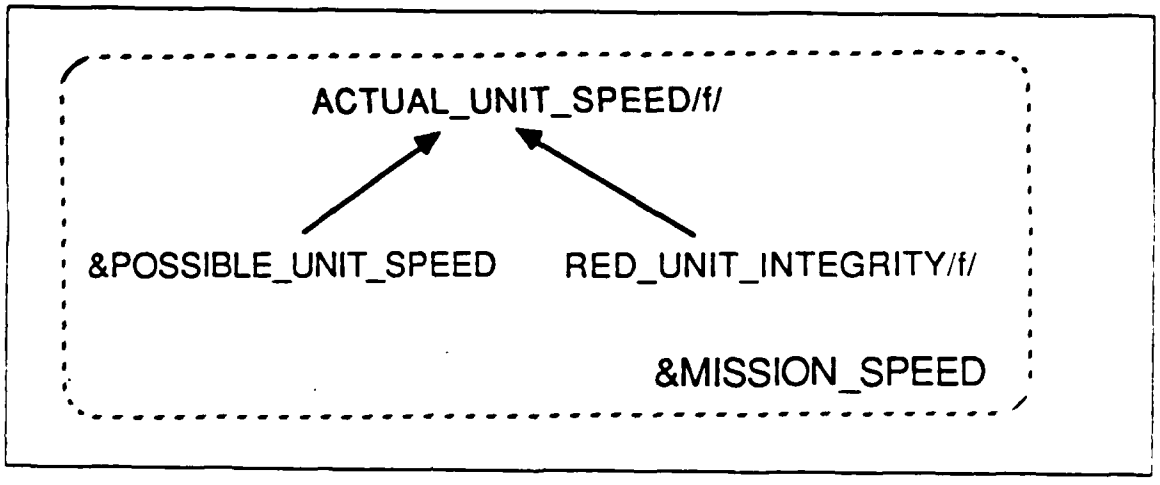


Figure B.19 Module MISSION_SPEED.

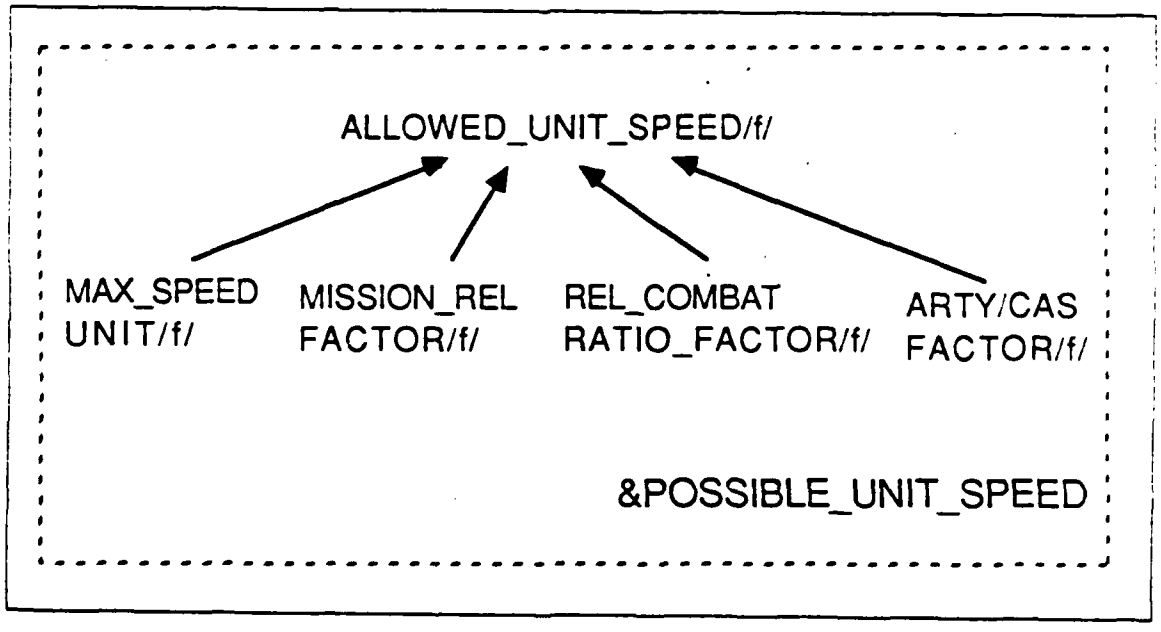


Figure B.20 Module POSSIBLE_UNIT_SPEED.

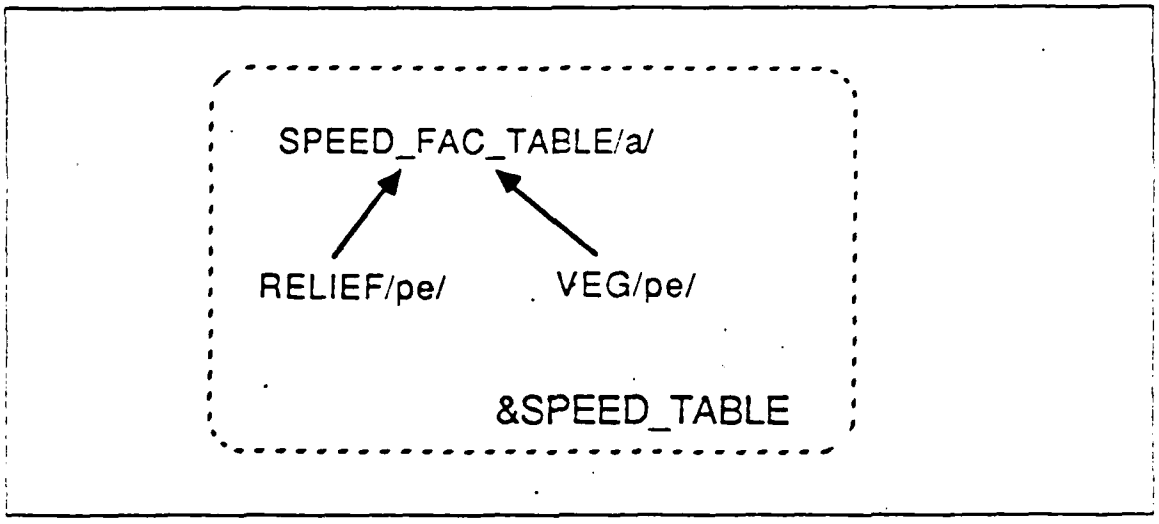


Figure B.21 Module SPEED_TABLE.

APPENDIX C
ELEMENTAL DETAIL TABLES

1. STEP 1

The first step of the elemental detail table structuring process is to generate a table structure for each genus in the model. The format for these tables is covered in Chapter IV of this thesis and on pages 2-46 - 2-52 of Reference 1.

IBL No table required

LOC_IBL
IBL || LOC_IBL

GRID_CELL
GRID_CELL || Interpretation

GRID_RELIEF
GRID_CELL || GRID_RELIEF

GRID_VEG
GRID_CELL || GRID_VEG

ROADS_AXIAL
GRID_CELL || ROADS_AXIAL

ROADS_LATERAL
GRID_CELL || ROADS_LATERAL

LOC_GRID_CELL
GRID_CELL || LOC_GRID_CELL

RELIEF
RELIEF || interpretation

VEG
VEG || interpretation

LARGE_UNIT
LARGE_UNIT || Interpretation

LOC_LARGE_UNIT
LARGE_UNIT || LOC_LARGE_UNIT

LARGE_UNIT_TYPE
LARGE_UNIT || LARGE_UNIT_TYPE

COMMITTED
LARGE_UNIT || COMMITTED

MOTION
LARGE_UNIT || MOTION

ENGAGED
LARGE_UNIT || ENGAGED

INFIGHT
LARGE_UNIT || INFIGHT

DIST_RAB_RMBIER
LARGE_UNIT || DIST_RAB_RMBIER

MIN_DIST
LARGE_UNIT || MIN_DIST

MOVING_MIN
LARGE_UNIT || MOVING_MIN

ORDERS
LARGE_UNIT || ORDERS

DESTINATION
LARGE_UNIT || DESTINATION

MISSION
LARGE_UNIT || MISSION

MISSION_CHANGE
LARGE_UNIT || MISSION_CHANGE

GIVEN_ORDERS
LARGE_UNIT || GIVEN_ORDERS

SPEED_FAC_TABLE
RELIEF, VEG || SPEED_FAC_TABLE

SPEED_FAC_CELL
GRID_CELL, RELIEF, VEG || SPEED_FAC_CELL

SPEED_FAC_AXIAL
GRID_CELL || SPEED_FAC_AXIAL

SPEED_FAC_LATERAL
GRID_CELL || SPEED_FAC_LATERAL

ROAD_SPEED_FAC
GRID_CELL, LARGE_UNIT || ROAD_SPEED_FAC

COM_SPEED_FAC_CELL
GRID_CELL, LARGE_UNIT || COM_SPEED_FAC_CELL

WEAPON
WEAPON || Interpretation

WEAPON_TYPE
WEAPON || WEAPON_TYPE

WEAPON_RANGE
WEAPON || WEAPON_RANGE

WEAPON_LIST
WEAPON, LARGE_UNIT ||

%AVAIL_WEAPON
WEAPON, LARGE_UNIT || %AVAIL_WEAPON

%AMMO_WEAPON
WEAPON, LARGE_UNIT || %AMMO_WEAPON

INFIGHT_WEAPON
WEAPON, LARGE_UNIT || INFIGHT_WEAPON

SMALL_UNIT
SMALL_UNIT || Interpretation

LOC_SMALL_UNIT
SMALL_UNIT || LOC_SMALL_UNIT

SMALL_UNIT_TYPE
SMALL_UNIT || SMALL_UNIT_TYPE

ASS_UNIT
SMALL_UNIT, LARGE_UNIT ||

TARGET_LIST
SMALL_UNIT, LARGE_UNIT ||

ALIVE_TARGET
SMALL_UNIT, LARGE_UNIT || ALIVE_TARGET

INFIGHT_TARGET
SMALL_UNIT, LARGE_UNIT || INFIGHT_TARGET

CALC_DIRECTION
LARGE_UNIT || CALC_DIRECTION

DIRECTION
LARGE_UNIT || DIRECTION

MAX_SPEED_UNIT
LARGE_UNIT || MAX_SPEED_UNIT

MISSION_REL_FACTOR
LARGE_UNIT || MISSION_REL_FACTOR

REL_COMBAT_RATIO_FACTOR
LARGE_UNIT, WEAPON || REL_COMBAT_RATIO_FACTOR

ARTY_CAS_FACTOR
LARGE_UNIT, WEAPON || ARTY_CAS_FACTOR

ALLOWED_UNIT_SPEED
LARGE_UNIT || ALLOWED_UNIT_SPEED

RED_UNIT_INTEGRITY
LARGE_UNIT, WEAPON, SMALL_UNIT || RED_UNIT_INTEGRITY

ACT_SPEED_UNIT
LARGE_UNIT || ACT_SPEED_UNIT

2. STEP 2

The second step of the elemental detail table structuring process deals with genera which used the functional or multi-valued dependencies in their generic calling sequences. [Ref. 1: pg. 2-47] In the case of the ONEC model these functional and multi-valued options were not used, so the second step of this process is not needed.

3. STEP 3

The third, and final step, in the elemental detail table structuring process is to combine as many of the tables as possible. Tables may be joined when they have identical stubs. The stubs must be identical for both the column headings and the rows. The identical column headings are easy to check by looking at the table structures from the first step. The identical row entries are harder to check. For this information it is necessary to check the index set statements of the respective genera. If they are identical then the row entries will be identical. An easier way to think of this is to consider that all tables with identical keys can be joined.

LOC_IBL
IBL || LOC_IBL

GRID_CELL
GRID_CELL || Interp, GRID_RELIEF, GRID_VEG,
ROADS_AXIAL, ROADS_LATERAL, LOC_GRID_CELL,
SPEED_FAC_AXIAL, SPEED_FAC_LATERAL

RELIEF
RELIEF || interpretation

VEG
VEG || interpretation

LARGE_UNIT
LARGE_UNIT || Interp, LOC_LARGE_UNIT, LARGE_UNIT_TYPE,
COMMITTED, MOTION, ENGAGED, INFIGHT, ORDERS,

DESTINATION, MISSION, MISSION_CHANGE, GIVEN_ORDERS,
CALC_DIRECTION, DIRECTION, MAX_SPEED_UNIT,
MISSION_REL_FACTOR, ALLOWED_UNIT_SPEED,
ACT_SPEED_UNIT

DIST_RAB_RMBIER
LARGE_UNIT || DIST_RAB_RMBIER

MIN_DIST
LARGE_UNIT || MIN_DIST, MOVING_MIN

SPEED_FAC_TABLE
RELIEF, VEG || SPEED_FAC_TABLE

SPEED_FAC_CELL
GRID_CELL, RELIEF, VEG || SPEED_FAC_CELL

ROAD_SPEED_FAC
GRID_CELL, LARGE_UNIT || ROAD_SPEED_FAC, COM_SPEED_FAC_CELL

*The correct table name is not clear, so the first name in the genus paragraphs was used.

WEAPON
WEAPON || Interp, WEAPON_TYPE, WEAPON_RANGE

WEAPON_LIST
WEAPON, LARGE_UNIT || %AVAIL_WEAPON, %AMMO_WEAPON,
INFIGHT_WEAPON

SMALL_UNIT
SMALL_UNIT || Interp, LOC_SMALL_UNIT, SMALL_UNIT_TYPE

ASS_UNIT
SMALL_UNIT, LARGE_UNIT ||

TARGET_LIST
SMALL_UNIT, LARGE_UNIT || ALIVE_TARGET, INFIGHT_TARGET

*These two tables, ASS_UNIT and TARGET_LIST are not joined because although the generic calling sequence is the same the index set statement is not.

REL_COMBAT_RATIO_FACTOR
LARGE_UNIT, WEAPON || REL_COMBAT_RATIO_FACTOR, ARTY_CAS_FACTOR

RED_UNIT_INTEGRITY
LARGE_UNIT, WEAPON, SMALL_UNIT || RED_UNIT_INTEGRITY

LIST OF REFERENCES

1. Geoffrion, A.M., *Structured Modeling*, Graduate School of Management, University of California, Los Angeles, CA, January 1985.

2. Army Training and Doctrine Command Systems Analysis Activity Technical Memorandum 3-78, *Command, Control, Communications and Combat Effectiveness Model Documentation Design Report*, Chapter 5, October 1978.

3. Geoffrion, A.M., *An Introduction to Structured Modeling*, Working Paper No. 338, Graduate School of Management, University of California, Los Angeles, CA, June 1986.

4. Geoffrion, A.M., *A Structured Modeling Representation of the GAMS Static Model of the Mexican Steel Industry*, Graduate School of Management, University of California, Los Angeles, CA, August 1986.

5. Geoffrion, A.M., *Hammer and McLeod's Semantic Database Model, Including Their Tanker Monitoring Database*, Unpublished paper, Graduate School of Management, University of California, Los Angeles, CA, 10 March 1986.

6. Geoffrion, A.M., *Comments on Smith and Smith*, Unpublished paper, Graduate School of Management, University of California, Los Angeles, CA, 13 March 1986.

7. Geoffrion, A.M., *Modeling Approaches and Systems Related to Structured Modeling*, Draft Working Paper No. 339, Graduate School of Management, University of California, Los Angeles, CA, July 1986.

8. Army Training and Doctrine Command Systems Analysis Activity TRASANA Program Summary Document, *FOURCE - Command, Control, Communications, and Combat Effectiveness*, 8 March 1985.

9. Geoffrion, A.M., *Syntax for Generic Rules*, Graduate School of Management, University of California, Los Angeles, CA, September 1986.

10. Geoffrion, A.M., *Index Set Grammar*, Graduate School of Management, University of California, Los Angeles, CA, September 1986.

11. Geoffrion, A.M., *Syntax for Generic Range Statements*, Graduate School of Management, University of California, Los Angeles, CA, October 1986.

12. Dolk, D.R., "Structured Modeling and Model Management: The Role of an Information Resource Dictionary System," Draft Paper, Department of Adminstrative Sciences, Naval Postgraduate School, Monterey, CA, Not Dated.

13. Geoffrion, A.M., *Modeling Categorization Hierarchies*, Informal Note, Graduate School of Management, University of California, Los Angeles, CA, 28 January 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Daniel R. Dolk Department of Adminstrative Sciences Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Prof. A. M. Geoffrion Graduate School of Management University of California Los Angeles, CA 90024	1
5.	Capt David J. Patrick SIO SYPC Peterson AFB, CO 80914-5001	3
6.	AFIT CISK Wright-Patterson AFB, OH 45433-6583	1
7.	Prof. Jeff Kottemann College of Business Administration Dept. of Decision Sciences 2404 Maile Way Honolulu, HI 96822	1
8.	U.S. Army TRADOC Systems Analysis Activity Attn: ATOR-TCA (Ben Morgan) White Sands Missile Range, NM 88002-5502	3

END

8-87

DTIC